AFRL-RB-WP-TR-2011-3022

RAPID PROPELLANT LOADING APPROACH EXPLORATION

Gregory Moster
Systems Integration Branch
Structures Division

NOVEMBER 2010
Final Report

AIR FORCE RESEARCH LABORATORY
AIR VEHICLES DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

[illegible]

GREGORY MOSTER, Program Engineer
Systems Integration Branch
Structures Division

[illegible]

Aaron Klosterman, Chief
Systems Integration Branch
Structures Division

[illegible]

DAVID M. PRATT, PhD
Technical Advisor
Structures Division

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YY)* November 2010	**2. REPORT TYPE** Final	**3. DATES COVERED *(From - To)*** 11 February 2008 – 15 November 2010

4. TITLE AND SUBTITLE RAPID PROPELLANT LOADING APPROACH EXPLORATION	**5a. CONTRACT NUMBER** In-house
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 0602201
6. AUTHOR(S) Gregory Moster	**5d. PROJECT NUMBER** 2401
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER A0FN0A
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Systems Integration Branch Structures Division Air Force Research Laboratory, Air Vehicles Directorate Wright-Patterson Air Force Base, OH 45433-7542 Air Force Materiel Command, United States Air Force	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFRL-RB-WP-TR-2011-3022
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Air Vehicles Directorate Wright-Patterson Air Force Base, OH 45433-7542 Air Force Materiel Command United States Air Force	**10. SPONSORING/MONITORING AGENCY ACRONYM(S)** AFRL/RBSI
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RB-WP-TR-2011-3022

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
PAO Case Number: 88ABW-2011-2190; Clearance Date: 14 Apr 2011.

This report was co-authored in collaboration with the NASA Kennedy Space Center.

14. ABSTRACT
This effort addresses two aspects of Reusable Military Launch Systems (RMLS). The first aspect is managing ground operations, and the second is comparing the impact upon ground operations of three configuration options. Ground operations management was addressed through a series of studies performed by the Air Force Institute of Technology (AFIT). The configuration options included 1) using a common bulkhead to separate the main liquid oxygen and kerosene main propellant tanks, 2) using a separate bulkhead to separate the main liquid oxygen and kerosene main propellant tanks, and 3) adjustments enabling inverted entry flight operations. These configurations were addressed through analysis by Boeing using their in-house software.

15. SUBJECT TERMS
rapid propellant loading, cryogenic fluids, liquid oxygen

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Gregory Moster
Unclassified	Unclassified	Unclassified	SAR	88	**19b. TELEPHONE NUMBER** *(Include Area Code)* N/A

TABLE OF CONTENTS

List of Figures

Figure **Page**

1. BACKGROUND

This joint AFRL and NASA research project was initiated in 1 March 2007 to begin addressing the technology and capability deficiencies in reducing the call-up time (time from mission notification until launch) to support the Prompt Global Strike (PGS) and on-orbit reconstitution missions. One of the critical call-up issues is loading cryogenic propellants (e.g. liquid oxygen).

Currently loading cryogenic propellants requires special operations and equipment with highly trained personnel to prevent vehicle damage and personnel injuries. Maintaining the required skill levels of these people are a key aspect of this process. The people must recognize problems when they arise and make critical decisions. These decisions not only affect vehicle and the people working near the vehicle but this mission as well. One way to reduce dependency upon critical personnel and enable rapid and consistent decisions is to create an automated system with built-in health and situation management. This would enable the personnel to connect the equipment and empower the system to perform the propellant loading operation with minimal oversight.

Funding for this project was provided by a combination of AFRL and NASA Kennedy Space Center.

2. APPROACH

The project goal was to develop technology to load cryogenic propellants rapidly, with a minimum number of console operators whose engineering skills and cryogenic experience are limited, such as a USAF non-commissioned officer (NCO). To accomplish this intelligent monitoring and diagnostics technology for cryogenic propellant loading using model-based reasoning software technology was selected. Model-based reasoning (MBR) is an innovative way of determining the dynamic state of a process by examining all available sensors instead of just one or a few. MBR also has the capability to determine the health of the system and predict degradations. Two NASA-developed MBR systems, Knowledge-based Autonomous Test Engineer (KATE) and Hybrid Diagnostic Engine (HyDE), were used to explore this capability. KATE and HyDE models described the physics of the cryogenic loading process and dynamically identified nominal and abnormal behaviors.

KATE and HyDE models incorporate cryogenic technical and operations knowledge allowing nearly autonomous system operation with minimal supervision. The MBR approach provided detection of cryogenic loading anomalies, isolation and identification of equipment faults, and recovery from failed instrumentation and components. KATE and HyDE software incorporated a physics-based simulation of vehicle propellant tanks, valves, ground loading piping and components. The software included a "reasoner" that compares the behavior of simulated and actual propellant-loading components to determine logically which measurements or components may have failed in the physical system to account for the symptoms that the control system is seeing.

The NASA software simulation final report is attached in Appendix A and a paper in Appendix B.

An additional objective was to load a LOX tank with LN2 using the software created above to validate the simulation results. However, an accident at KSC impacted this program and limited the LN2 loading experiment to trying to find ways to reduce the about of fluid boiling off and venting out of the tank as a gas. This would be beneficial because it reduces the about of fluid required to fill the tank, enables a smaller diameter tank vent line, lowers the pressure build-up during the loading process, and conditions the tanks for a rapid fill. The approaches selected focused upon spraying the cryogenic fluid inside of the tank using a shower head like devise to chill the interior tank walls down.

The LN2 loading experiment final report is attached in Appendix C.

The project leveraged and synergized with work performed on the Rapid Propellant Loading (RPL) test bed at NASA's Cryogenics Test Laboratory.

3. RESULTS

The software Block Diagram shown in figure 1 depicts the launch site software architecture developed for the MBR system. The KATE and HyDE software runs in the RPL Diagnostics Console. Real world data is supplied to the system via LabView hardware and software. The Internet Communications Engine (Ice) publish/subscribe architecture provides a connection between the data source – live, simulated or recorded and the MBR application in the diagnostics console. The RPL OPS console communicates between the USAF mission operations center and the launch site. Advanced Diagnostics and Prognostics Test-bed (ADAPT) API is NASA software that enables communication with a variety of diagnostic software.

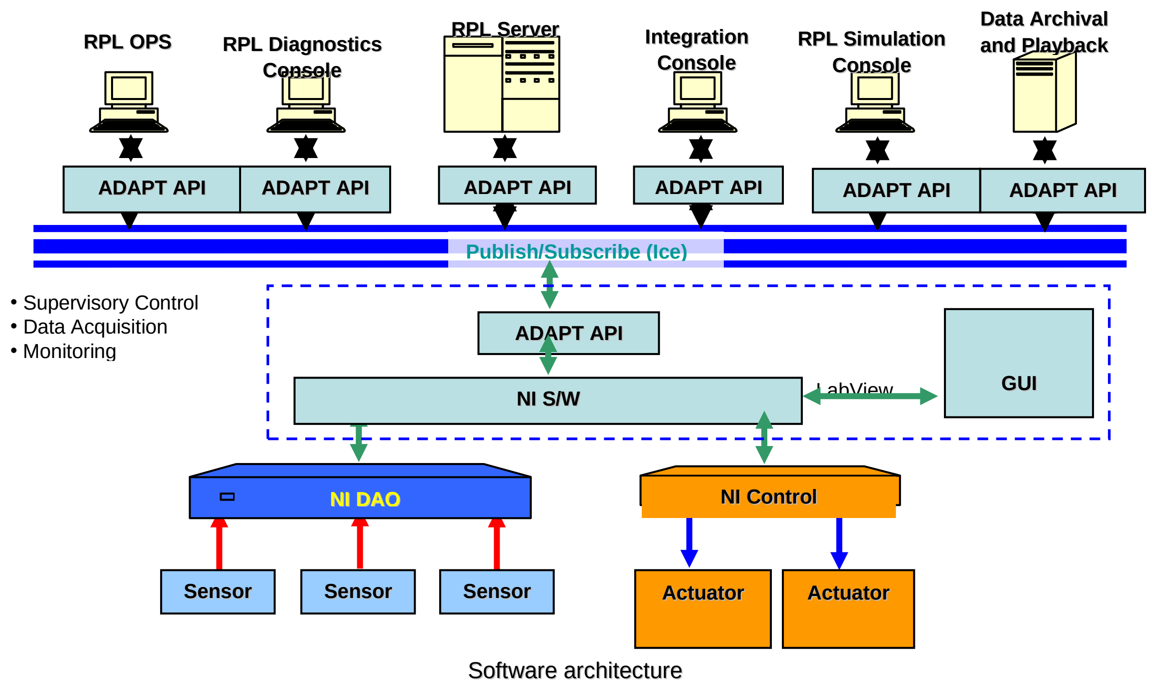


Figure 1
Software Block Diagram

A "Mini-model" simulation was developed to test the MBR software in two versions, Excel and Matlab. For the final demonstrations, Matlab/Simulink data from the mini-model simulator was supplied to KATE and HyDE for diagnosis tests by means of text files. A schematic diagram of the mini model is shown in figure 2. The model consists of a cryogenic storage tank, two pumps, valves and flow/pressure measurements, and a simulated vehicle tank for propellant loading. The tank, piping and instrumentation sizes were chosen to correspond to the Rapid Propellant Loading (RPL) test bed at NASA's Cryogenics Test Laboratory.

A large storage tank holds the propellant. A pump is used to create a pressure differential between the two tanks in order to move fluid from the storage tank to the vehicle tank. Flow (F), pressure (P), level (L), and valve position (V) sensors are included, as denoted in the above schematic. Note that this simulation represents a simplified propellant loading model. In particular, we neglect boiling/condensation processes, heat flow, and temperatures (a constant gas temperature is assumed).

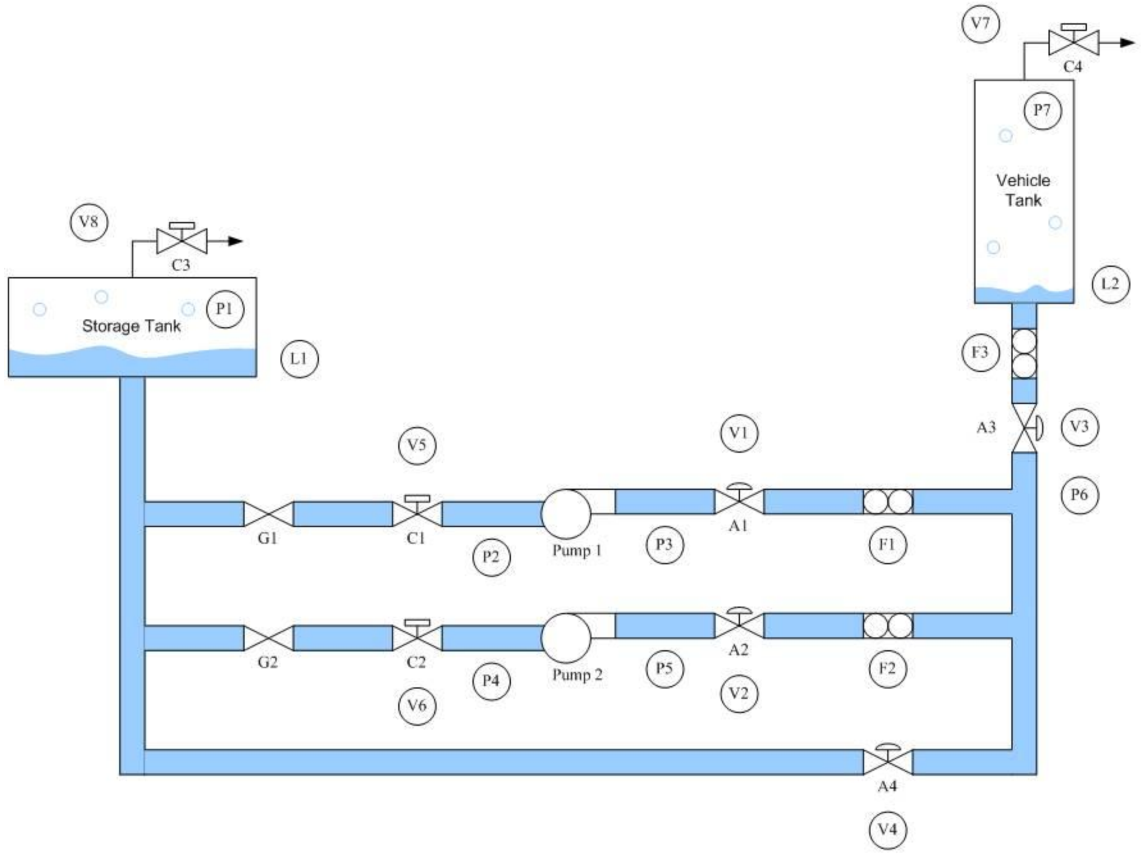


Figure 2
Mini Model

The software team developed a Mini Model use case for nominal propellant loading showing how cryogenics would flow through the mini-model during normal operation and three use cases illustrating how the control system should respond to typical component faults. The three fault use case scenario's are 1) faulty valves, 2) failed flow sensors, and 3) vent valve failure. The Top Level Simulation Mini Model is shown in figure 3.

The values of the inputs are determined by the current filling stage, specified by a set of parameters indicating when the various stages begin. The 'LN2Control' block, which calls LN2Control.m, determines the stage. The set of stages are:

1. *Initialization*: All manual and discretely-controlled valves are set open, and all flow control valves are set at 50%. The pumps are turned off.
2. *Slow Fill:* The pumps are set to 300 RPM. Slow fill begins when the simulation time is 'tSlow'.
3. *A4 Test*: Valve A4 is set to 10%. This occurs when the simulation time is 'tA4', which should be greater than 'tSlow'.

4. *Fast Fill*: The pumps are set to 2500 RPM. Vent valve C4 is placed into autonomous mode, where it will open at 7 psig and close at 2 psig. Fast fill begins when the simulation time is ‘tFast’, which should be greater than ‘tA4’.
5. *Topping*: Pump speeds are lowered, A4 is partially opened more than 10%, and A3 is controlled to maintain F3 at 20 GPM. Topping occurs when the tank level reaches 980 gallons, which should occur after ‘tFast’.
6. *Replenish*: A4 is opened to a greater extent. Vent valve C4 is set open. Replenish begins when the tank level reaches 1000 gallons.
7. *Drainback*: All flow control valves are set to 100%. The pumps are turned off. Drainback occurs when the simulation time is ‘tDrain’, which should be after L2 reaches 1000 gallons.
8. *End*: Vent valve C4 is closed. This occurs when L2 reaches 10 gallons.

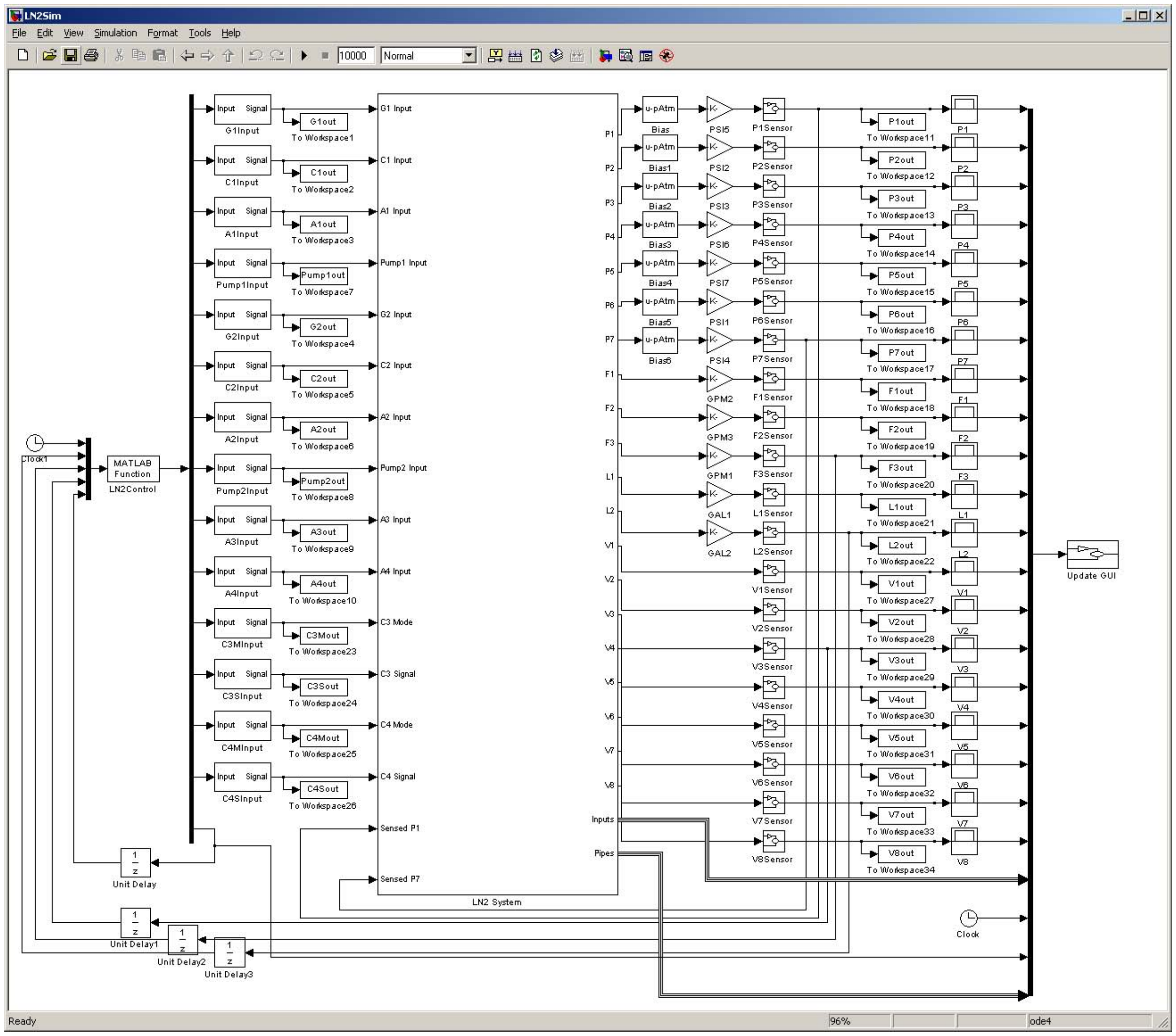
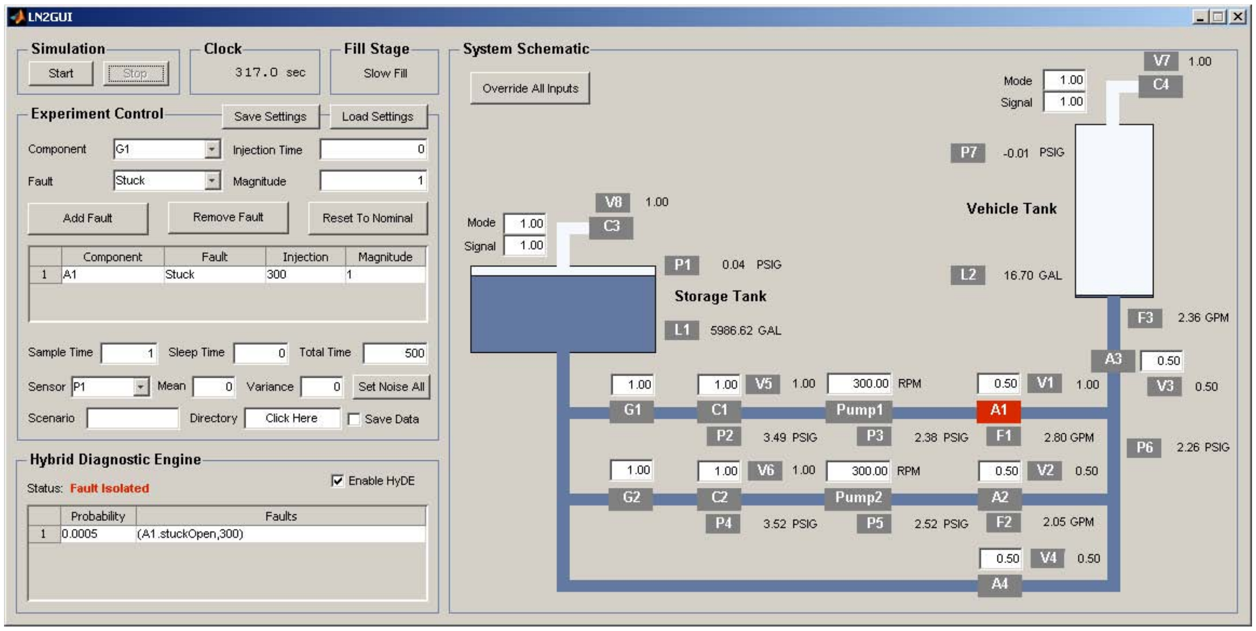


Figure 3
Top Level Simulation Mini Model

The mini-model KATE model was constructed as a group of KATE objects Knowledge Base (KB). The KB is made up of two parts: a) a library of components that are compiled into the C++ system, each library object has its input names and output function (a single output function for each component is used) and b) a "flatfile" textual description of the model or KB specifying library components and how things are connected together as well as parameter values.

Diagnostics are performed using the HyDE software. HyDE uses fluid flow, fluid pressure, and fluid level sensors for diagnostics in these simulations. Valve position and other sensors are not used.

In the first scenario, Flow Control Valve A1 was stuck open during the slow fill stage as shown in figure 4 screenshot. The system begins in the initialization phase, and begins slow fill at 100 seconds. Flow sensor F1 measures the flow through the first transfer line. An increase in the flow is observed at this time due to the change in control inputs. At 300 seconds, a fault is injected. A sharp increase in F1 is visible (a dotted black line in the simulation plot indicates the nominal value at this point, and the solid blue line indicates the sensor output). HyDE immediately detects and isolates the fault at this time. At around 330 seconds, the operator reconfigures to resume a nominal flow rate through the transfer line by reducing the Pump 1 input to 275 RPM.



Clock: 317.0 sec

Fill Stage: Slow Fill

Component: G1 Injection Time: 0

Fault: Stuck Magnitude: 1

	Component	Fault	Injection	Magnitude
1	A1	Stuck	300	1

Sample Time: 1 Sleep Time: 0 Total Time: 500

Sensor: P1 Mean: 0 Variance: 0

Scenario: Directory: Click Here Save Data

Status: Fault Isolated

☑ Enable HyDE

	Probability	Faults
1	0.0005	(A1.stuckOpen,300)

Mode 1.00 Signal 1.00 (C4); V7 1.00

P7 -0.01 PSIG

L2 16.70 GAL

F3 2.36 GPM

A3 0.50; V3 0.50

P6 2.26 PSIG

Mode 1.00 Signal 1.00 (C3); V8 1.00

P1 0.04 PSIG

L1 5986.62 GAL

G1 1.00; C1 1.00; V5 1.00; Pump1 300.00 RPM; A1 0.50; V1 1.00

P2 3.49 PSIG; P3 2.38 PSIG; F1 2.80 GPM

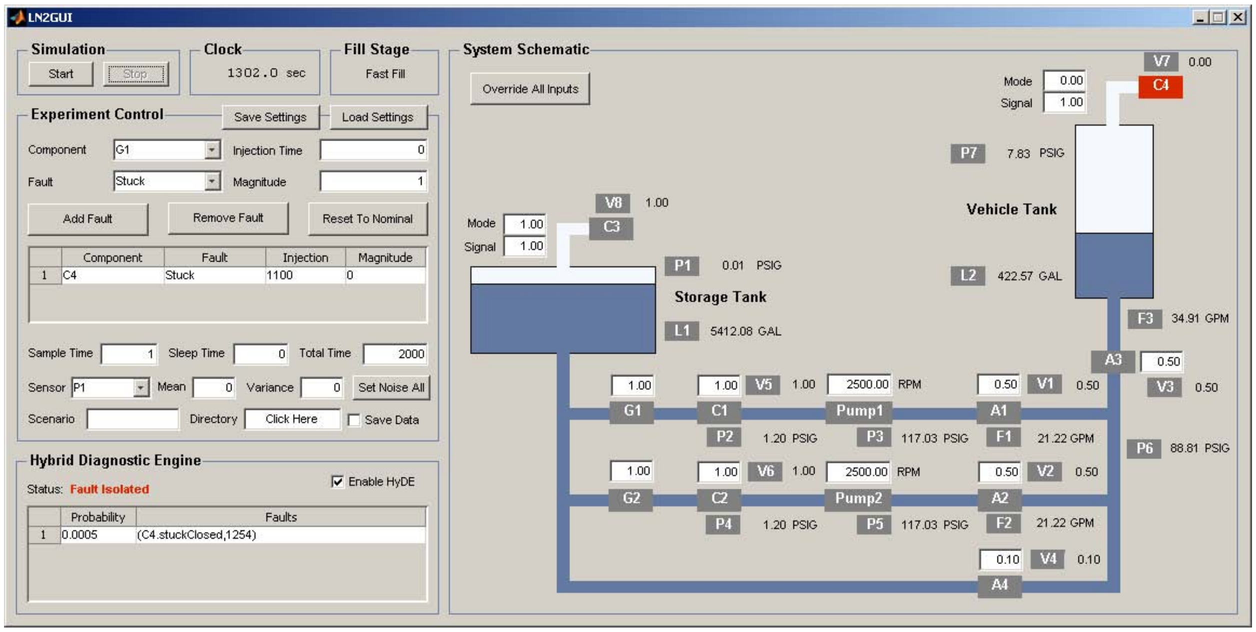
G2 1.00; C2 1.00; V6 1.00; Pump2 300.00 RPM; A2 0.50; V2 0.50

P4 3.52 PSIG; P5 2.52 PSIG; F2 2.05 GPM

A4 0.50; V4 0.50

Figure 4
Flow Control Valve Stuck Open Scenario

In the second scenario, the vehicle tank vent valve, C4, becomes stuck closed during the slow fill stage as shown in figure 5 screenshot. The system begins in the initialization stage and proceeds to slow fill at 100 seconds, and fast fill at 600 seconds. As the vehicle tank fills, the ullage pressure, measured by P7, begins to increase. In nominal operation, the vent valve C4 automatically opens when the ullage pressure reaches 7 psig, and closes when the ullage pressure reaches 3 psig. In this way, it maintains the ullage pressure within safe levels. If the vent valve fails to open at the required time, the ullage pressure may build up to unacceptable levels and an abort will have to be issued. Therefore, it is important to quickly identify such failures. The vent valve C4 becomes stuck closed at 1100 seconds. At this time, it is supposed to remain closed, as the ullage pressure is not yet at 7 psig. At 1254 seconds, the ullage pressure begins to exceed 7 psig. HyDE immediately recognizes that C4 has failed. Around 1340 seconds, the operator initiates drain back to safe the system. The pumps are turned off and all valves are fully opened.


Clock: 1302.0 sec
Fill Stage: Fast Fill

Experiment Control

Component: G1 | Injection Time: 0
Fault: Stuck | Magnitude: 1

	Component	Fault	Injection	Magnitude
1	C4	Stuck	1100	0

Sample Time: 1 | Sleep Time: 0 | Total Time: 2000
Sensor: P1 | Mean: 0 | Variance: 0
Scenario: | Directory: Click Here | ☐ Save Data

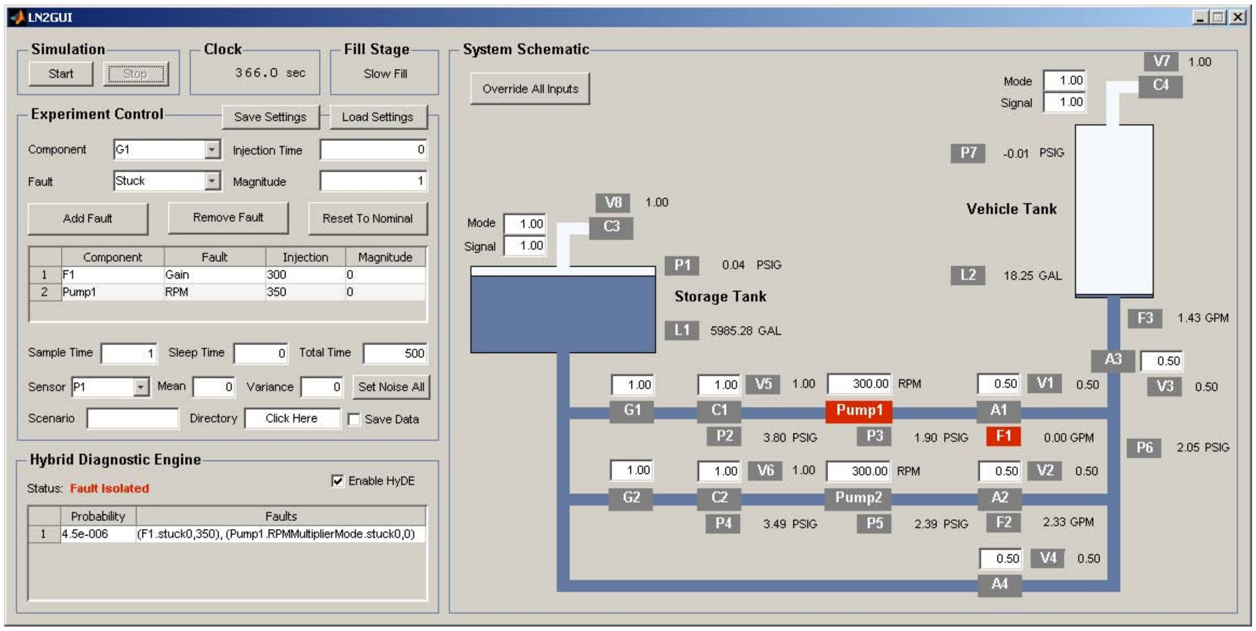
Hybrid Diagnostic Engine

Status: Fault Isolated | ☑ Enable HyDE

	Probability	Faults
1	0.0005	(C4.stuckClosed,1254)

Figure 5
Vehicle Tank Valve Stuck Open Scerario

In the third scenario, two faults occur as shown in figure 6 screenshot. The flow sensor (F1) fails followed by Pump 1 failure during the slow fill stage. The system begins in the initialization stage, and begins slow fill at 100 seconds. At 300 seconds, the sensor fails and returns only a value of zero. HyDE immediately determines that the flow sensor has failed in this way. At this point, only the remaining flow sensors, F2 and F3, are available for diagnosis. At 350 seconds, Pump 1 fails. HyDE immediately detects that something else has gone wrong and initially pinpoints a failure of A1. At the next time step, HyDE corrects its diagnosis and pinpoints Pump 1 as the second failure. (Note that if the pressure and level sensors are also used, HyDE will immediately diagnose Pump 1 as the second failure, and not A1.) At around 380 seconds, the operator reconfigures the system to regain nominal flow to the vehicle tank despite the failure of Pump 1. Valve A1 is closed, valve A2 is fully opened, and the RPM of Pump 2 is increased to 375. The flow to the vehicle tank is then restored to the nominal value of about 2.2 GPM.



Clock: 366.0 sec

Fill Stage: Slow Fill

Experiment Control

Component: G1   Injection Time: 0

Fault: Stuck   Magnitude: 1

	Component	Fault	Injection	Magnitude
1	F1	Gain	300	0
2	Pump1	RPM	350	0

Sample Time: 1   Sleep Time: 0   Total Time: 500

Sensor: P1   Mean: 0   Variance: 0

Scenario: Directory: Click Here   ☐ Save Data

Hybrid Diagnostic Engine

Status: Fault Isolated   ☑ Enable HyDE

	Probability	Faults
1	4.5e-006	(F1.stuck0,350), (Pump1.RPMMultiplierMode.stuck0,0)

Figure 6
Flow Sensor and Pump Failure Scenario

The NASA simulation final report is attached in Appendix A and paper in Appendix B.

The LN2 loading experiment is shown in Figure 7 and spray nozzle in Figure 8. Detailed test set-up and results are presented in Appendix C "Rapid Propellant Loading (RPL) Cryogenic Tanking Demonstration".

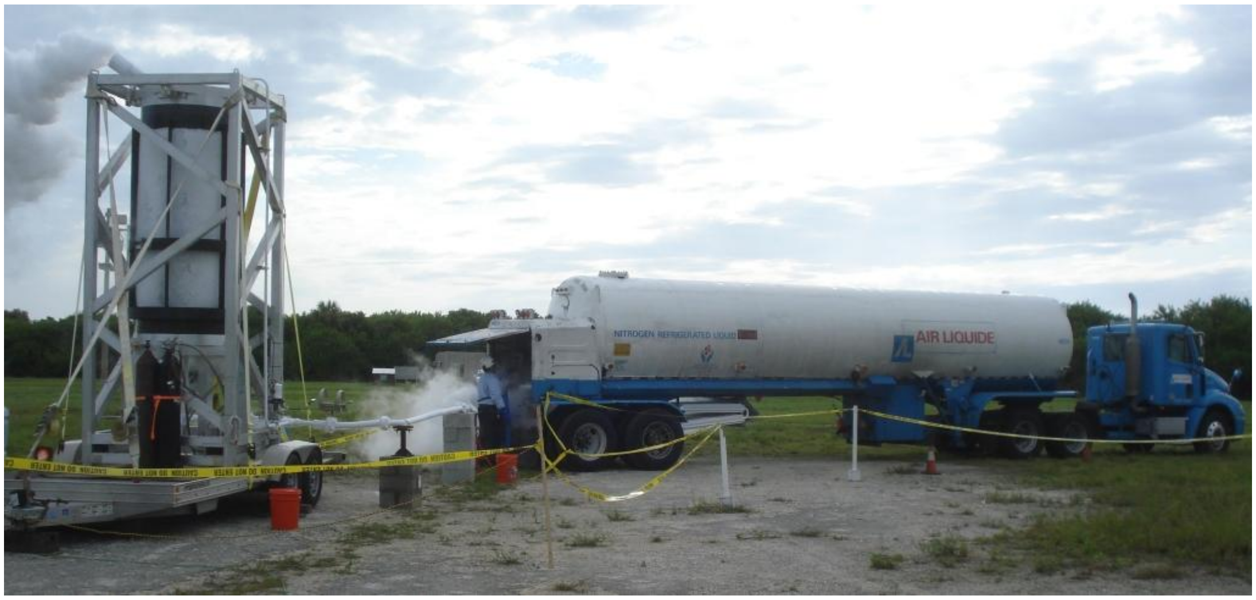


Figure 7
LN2 Loading Experiment

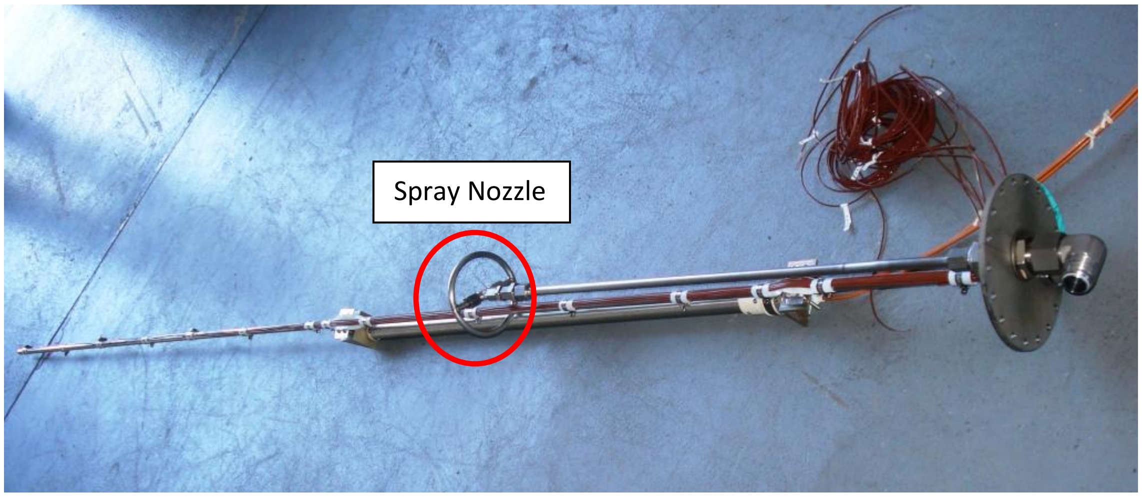


Figure 8
Spay Nozzle Test Hardware

4. CONCLUSIONS

The simulation results demonstrated that an integrated health and operations management system can enable safe rapid propellant loading operations with limited operator knowledge and involvement. A single person loading the propellants with back-shop support to perform maintenance and repairs appears to be feasible. The next step would be to expand the scale and scope of the simulation and experimentally validate the results.

Unfortunately, cost, test site, and safety constraints levied upon the LN2 loading experiment restricted the effort sufficiently that the results are not sufficient to assess the spraying techniques attempted. Another experiment with greater latitude (e.g. tank pressure, tank insulation, & temperature/fluid level measurements) should be performed to evaluate this technology.

APPENDIX A

Rapid Propellant Loading Software Final Report

October 6, 2009

Prepared by:

Charles Goodrich

ASRC Aerospace Corporation
Kennedy Space Center, FL 32899
321-867-6956

Contributions by:

David Richter

ASRC Aerospace Corporation
Kennedy Space Center, FL 32899
321-867-2032

Sriram Narasimhan

University of California Santa Cruz USA

Matt Daigle

University of California Santa Cruz USA

Submitted to:

Robert G. Johnson

NASA-KSC, NE-F6
Kennedy Space Center, FL 32899
(321) 867-7373

Gregory Moster

AFRL/RBSD
Wright-Patterson AFB, OH

Barbara L. Brown

ARC
Ames Research Center, CA
(321)867-1720

Jose M. Perotti

NASA-KSC, KT-C
Kennedy Space Center, FL 32899
(321)867-6746

CONTENTS

FIGURES

A1. BACKGROUND

The Air Force Research Laboratory (AFRL) in Dayton, OH is teaming with aerospace contractors in a program to identify and address technology drivers for responsive, on-demand access to, through and from space. AFRL requirements call for on-demand, flexible, and cost-effective operations using reusable rockets. Operability enhancement goals set by AFRL include rapid turnaround (< 4 hours), lower operations cost, and much higher vehicle and ground support equipment reliability.

A2. DESCRIPTION OF GROUND PROCESSING PROBLEMS AND GOALS

Launch vehicle cryogenic propellant loading is one of the most complex and hazardous operations performed during launch processing. This project demonstrated intelligent monitoring and diagnostics technology for cryogenic propellant loading using model-based reasoning software. Model-based reasoning (MBR) is an innovative way of determining the dynamic state of a process by examining all available sensors instead of just one or a few. MBR also has the capability to determine the health of the system and predict degradations. Two NASA-developed MBR systems, Knowledge-based Autonomous Test Engineer (KATE) and Hybrid Diagnostic Engine (HyDE), were used to explore this capability. KATE and HyDE models described the physics of the cryogenic loading process and dynamically identified nominal and abnormal behaviors. The technology was developed and demonstrated under a Space Act Agreement between KSC and AFRL.

A3. APPLICABILITY TO AFRL REQUIREMENTS

The project goal was to develop technology to load cryogenic propellants rapidly, with a minimum number of console operators whose engineering skills and cryogenic experience are limited, such as a USAF non-commissioned officer (NCO). KATE and HyDE models incorporate cryogenic technical and operations knowledge allowing nearly autonomous system operation with minimal supervision. The MBR approach provided detection of cryogenic loading anomalies, isolation and identification of equipment faults, and recovery from failed instrumentation and components. KATE and HyDE software incorporated a physics-based simulation of vehicle propellant tanks, valves, ground loading piping and components. The software included a "reasoner" that compares the behavior of simulated and actual propellant-loading components to determine logically which measurements or components may have failed in the physical system to account for the symptoms that the control system is seeing.

The project leveraged and synergized with work performed on the Rapid Propellant Loading (RPL) test bed at NASA's Cryogenics Test Laboratory.

A4. RELEVANCE TO OTHER AFRL TECHNOLOGY NEEDS

KATE and HyDE software is applicable to a "Reasoner" capability needed for other reusable rocket technology including active thermal control, purge and pressurization, avionics checkout, electrical distribution, and the rocket's main propulsion system. The Reasoner thus provides Fault Detection, Isolation and Recovery (FDIR) for the launch vehicle and ground support equipment.

Software Team

ASRC

- Charlie Goodrich – ASRC Software Lead
- David Richter
- Bradley Burns

NASA (KSC)

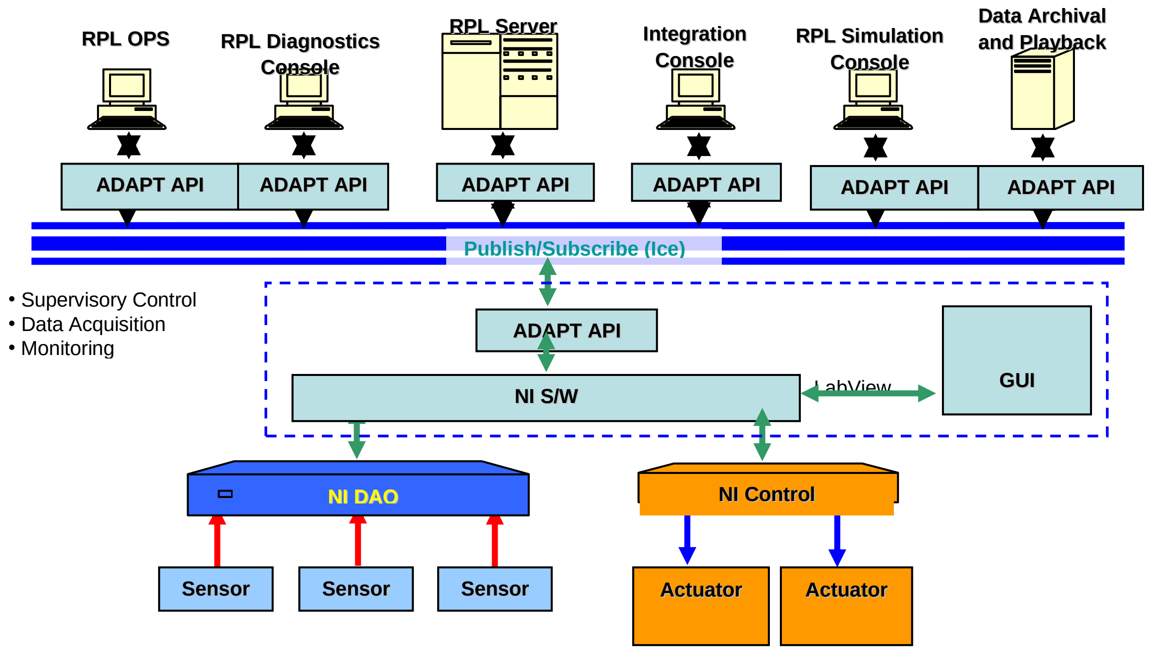
- Jan Lomnes – Project Manger
- Robert Johnson – NASA Hardware Lead
- Jose Perotti
- Robert Ferrell
- Jared Sass

NASA (ARC)

- Barbara Brown – NASA Software Lead
- Ann Patterson-Hine
- Sriram Narasimhan - University of California, Santa Cruz
- Matthew Daigle – UARC

A5. SOFTWARE BLOCK DIAGRAM

The following diagram shows the launch site software architecture developed for the MBR system. The KATE and HyDE software runs in the RPL Diagnostics Console. Real world data is supplied to the system via LabView hardware and software. The Internet Communications Engine (Ice) publish/subscribe architecture provides a connection between the data source – live, simulated or recorded and the MBR application in the diagnostics console. The RPL OPS console communicates between the USAF mission operations center and the launch site. ADAPT API is NASA software that allows The Advanced Diagnostics and Prognostics Testbed (ADAPT) to communicate with a variety of diagnostic software.



Software architecture

Figure A-1
Software Block Diagram

Because of limited funds and time for hardware development the decision was made to test KATE/HyDE in simulated mode only.

A6. MINI MODEL

The "Mini-model" simulation was developed to test the MBR software in two versions, Excel and Matlab. For the final demonstrations, Matlab/Simulink data from the mini-model simulator was supplied to KATE and HyDE for diagnosis tests by means of text files. A schematic diagram of the mini model is shown below:

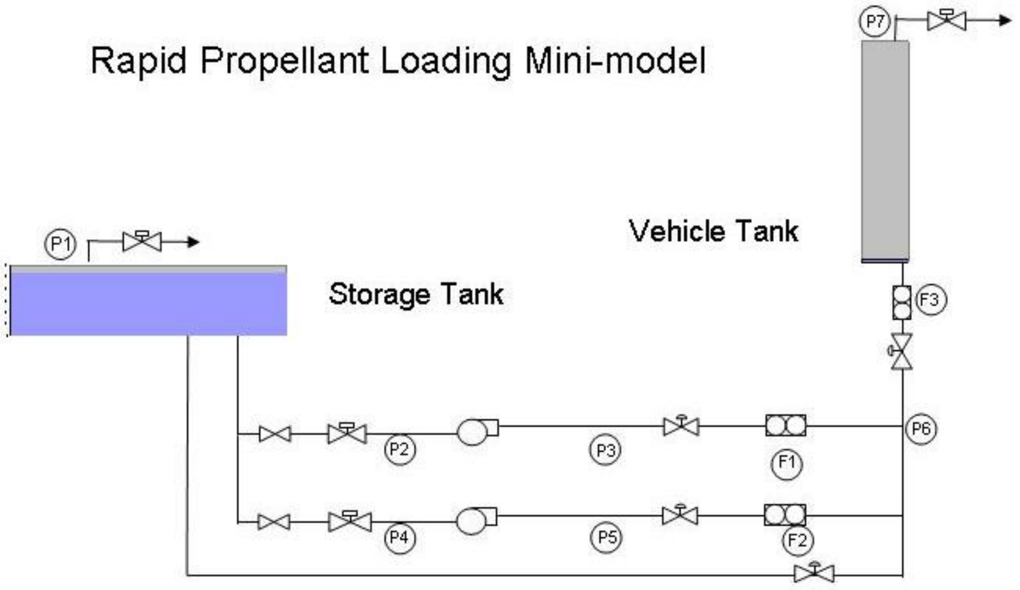


Figure A-2
Mini Model

The model consists of a cryogenic storage tank, two pumps, valves and flow/pressure measurements, and a simulated vehicle tank for propellant loading. The tank, piping and instrumentation sizes were chosen to correspond to the Rapid Propellant Loading (RPL) test bed at NASA's Cryogenics Test Laboratory.

<u>Testing Strategy for FDIR applications using the mini-model and use cases</u>
The software team developed a use case for nominal propellant loading showing how cryogenics would flow through the mini-model during normal operation. In addition, the team developed three additional use cases illustrating how the control system should respond to component faults that might occur. The three fault use cases are 1) faulty valves; 2) failed flow sensors; and 3) vent valve failure.

A7. KATE-SPECIFIC ITEMS

KATE C++ Code

KATE-C is a FDIR application developed at Kennedy Space Center in 1994. The software is written in C++. It has been recompiled using up-to-date compilers, state-of-the-art hardware, and the Linux operating system. KATE-C was tested using application knowledge bases from 1994. A KATE-C knowledge base for the mini-model was completed, and the system was demonstrated for the USAF in December 2008. A list of changes/fixes made to 1994 version is included in the installation instructions for KATE-C (below). A narrative for major revisions of the software made by the development team is part of the software repository that accompanies this report.

Instructions to compile and run KATE

The installation instructions for KATE (including running a mini-model failure scenario) are included with the software that accompanies this report. The software was developed under the version control system known as Subversion. It is provided in two forms. The first is an archive of the Subversion repository called rpl.repo.tar.gz in the KATE directory. The second is a tarball of the current "working copy" of the repository called rpl.tar.gz also in the KATE directory. The installation instructions are included in the following paragraphs for convenience:

```
//********************************************
//**                                        **
//**  File:  README.install                 **
//**                                        **
//**  Original by D. Merchant circa 1994    **
//**  Updated by C. Goodrich and D. Richter **
//**  9-30-2009                             **
//**                                        **
//********************************************

Installation instructions for KATE*

Depending on how the KATE source is delivered, either unpack the source tarball
or retrieve the source from a previously-configured Subversion database.

KATE Development Environment:

Note that the original (1994) KATE software used the Motif graphic display
system. When KATE was updated in 2008-2009, developers used Open Motif. The home
page for the Open Motif development project is http://www.openmotif.org/ Motif
is an industry standard toolkit for UNIX systems and Open Motif, based on the
Motif source code, provides a freely available version for open source
developers.

The KATE update project in 2009 also used Red Hat Enterprise Linux as the
operating system platform for development.  The system was developed on a Dell
computer using a 64 bit Intel processor.  The Motif components of the Red Hat
development environment code were installed using yum, a software installation
tool for Red hat linux and Fedora Linux. Here are the Open Motif components:
openmotif.i386                          2.3.1-2.el5
openmotif.x86_64                        2.3.1-2.el5
openmotif-devel.i386                    2.3.1-2.el5
openmotif-devel.x86_64                  2.3.1-2.el5


1) Open a BASH shell and create the root KATE directory (hereafter called KATE)
   and change to that directory.
   a) Assuming the source is delivered as KATE.tar, move the tarball to the KATE
      directory and untar it: tar xvf KATE.tar
   b) Assuming the source resides in a Subversion database, issue the Subversion
      command to checkout the source: svn co https://128.217.72.10/svn/rpl/ice/trunk .
      (modify the URL and path as needed).

2) Create the following directories, needed in the compilation / linking process:
   KATE/sun/lib and KATE/sun/obj.

3) Create the /ice directory at root level and move the ice.hosts file into that
directory.
   Edit the file and add the local IP and hostname. For example here is a sample
ice.hosts
   file for a single machine named "katehost"

      #
      # Host Database For KATE
      #
      127.0.0.1 katehost

4) Edit the local .bashrc file and add the following definitions:
   export CPP=$(which g++)
   export ICE_DIR="${HOME}/KATE/"
   export ICE_SRC_DIR="${HOME}/KATE/src/"
   export ICE_MAC_DIR="${HOME}/KATE/sun/"
   export COMPILER="-g -Wno-deprecated -fno-operator-names"
   export PATH=$PATH:${HOME}/KATE/sun/exe:.

   a) You must redirect the CPP command to use g++, per the above environment
      command as the modern cpp command invokes the C preprocessor rather than
      the compiler.
   b) Ensure that you add the trailing slash to the directory paths.
   c) You will need to re-source the .bashrc file for the changes to take effect,
      i.e., in your home directory issue the command: source .bashrc

5) Change to the KATE/src directory and make the myMakeIt script executable:
   chmod a+x myMakeIt

6) Build the source by executing the myMakeIt script in the KATE/src directory:
   myMakeIt.
   If the source compiles with no problems you should see the following text:

   ************************************************************
   Compile complete
   See: Running ICE-C with the GUI in the README.install file
   ************************************************************

7) To run KATE, change to the KATE/sun/exe directory and issue the command: ui
   This will start KATE in the default mode with no windows open. To run KATE with
   a previously-saved startup script, issue the command: ui -uienv <script>
   (see the discussion of scripts under the topic "Running ICE-C (KATE) with the
    GUI" below)

* For historical reasons KATE is also sometimes referred to as ICE or ICE-C.

---------------------------------------------------------------------------

KATE directory hierarchy:

    KATE
        data (*.kpd - for KATE processed datafiles)
        models
            alo
            DB
            epdc
            fcl
            instr
            lox
            lsoc
            net
            pvd
            translator
        src
            adapt
            dm
            kb
                alo (water tanking system)
                common
                epdc
                fcl (freon coolant loop)
                lox (liguid oxygen)
                net
                ptrespository
                pvd (vent doors)
            kics
            reasoner
                diagnoser
                fault-detector
                monitor
                simulator
            translator
            ui
                ICONS
            utils
        sun
            exe
            lib
            obj

----------------------------------------------------------------------------

The KATE system is made up of four modules and a Knowledge Base (KB)

1) The User Interface module (UI) is an X-Window/MOTIF interface that displays
   values and provides control access to the other modules.

2) The Reasoning Module (RM) which, as the name suggests, does all the Model-based
   Reasoning (MBR) in the system.  The four sub-modules with the RM are the
   Monitor, Simulator, Fault Detector, and Diagnoser.

3) The Data Module (DM) which is responsible for getting data to the Reasoning
   module. There are currently two data providers: a Playback facility that
   enables a recorded data file to be played back, and an Ethernet data
   provider that allows KATE to connect to PC-GOAL via an ethernet
   connection. (Note that this second functionality is not available at
   this time.) The Data Module also includes the Record facility which
   allows recording of data from any data provider and a Build program for
   building PC-GOAL TCID files used to look up FD names with an address.

   The input to build is a TCID file and an FD list. This program assumes
   the filename extensions .nam, .bsd, and .bsi.

   (The TCID build facility developed in 1994 has not been updated to PCGOAL II.)

4) The KICS system is the KATE Interprocess Communication System that
   allows all the processes in the system to talk to each other.

5) The KB is made up of two parts:
   a) A library of components that are compiled into the system, each library
      object has its input names and output function (a single output
      function for each component is used).  The class libraries are broken up into
      application-specific files and located in KATE/src/kb/<application>,
      where <application> is fcl, lox, alo, etc. All application source
      libraries are compiled into one complete class library - so there can
      be no duplication of class names between applications.  The Rapid Propellant
      loading class libraries were included in the KATE/src/kb/alo directory.
   b) A "flatfile" textual description of the application domain specifying
      library components and how things are connected together as well as
      parameter values. See KATE/README.flatfile for a brief knowledge base
      generation guide.

---------------------------------------------------------------------------
```

```
Other files to be aware of

1) The GUI can be started with a -uienv option to allow the system to use
   a startup script, as previously mentioned: ui -uienv <script>

2) Browser description file. The GUI provides a tabular utility called
   the Browser for displaying objects and their model and hardware values
   along with the nomenclature if desired. A predefined list of objects to
   be displayed need to be in a browser file. This file can be read
   in by the startup script or manually by the user through the GUI.

   The Browser facility worked in the original Motif graphics
   implementation but has not been debugged for use in the Open Motif
   environment.

3) $(ICE_DIR)data/playback.kpd - holds the names and locations of playback files
   which is used only when running the text ui (pui.C) for playback.

4) $(ICE_SRC_DIR)make.macros - this file is read by all makefiles in the system
   and is used to set up preprocessor, compile and linker flags as well as set up
   the macros for the directories in the system and define the include paths.

   CPPLINK = -Bstatic  // Used for static linking (usually for delivery).

5) A file called archive.dat is created when the archiving option on the
   simulator is turned on by clicking the radio button under "Diagnoser
   Tools" in the reasoner GUI.  This file can then be replayed using the
   facility (even though the extension is not kpd).  Be careful to rename
   the file because if an archive.dat file exists it will get overwritten
   when archiving is turned on.

6) The file models.list is the list of KB flatfiles used by the rm-ui
   (the text user interface for the reasoner). The path for this file
   is currently hardcoded as "/ice"; see rm-ui.C to change it.

   Using the graphical user interface (GUI) it is not necessary to build a
   models.list file since the menu command: File ==> Load KB brings up a
   list of all files in the current directory with the .flatfile extension

7) The file $(ICE_SRC_DIR)utils/kate-globals.C defines the services that are
   available (client/server).  There were  five sets
   of communication services available in 1994, but this can be increased at
   any time.  This means that there can be five separate KATE instances running
   simultaneously (on the same or different hosts).

   Another place that path names are hard coded into the system is rui.C
   (the text interface for record). It currently has "/ice" as the default
   directory for its files.

   The Diagnoser does write out a temporary file during a diagnosis
   but this is written to /tmp and then is removed. Check work-bench.C
   in $(ICE_SRC_DIR)reasoner/diagnoser if you want to change that.

Note: the "kpd" extension stands for Kate Processed Data and should (though
not required) be the extension for all files that are to be played back to
KATE using the playback facility.

---------------------------------------------------------------------------

Running ICE-C (KATE) with the GUI:

   To run the Graphical User Interface:

   The graphical interface may be run with or without scripts. Scripts
   provide a shortcut to loading the knowledge base (flatfile) and various
   KATE graphical user interface modules such as Reasoner, Data Provider,
   Browser, Tree Display and Schematic Viewer. The GUI has a facility to
   save this manually-loaded window configuration in a script file so that
   the full configuration may be reloaded easily.
   The following example illustrates how to run the GUI with both methods.
   After the initial manual run, the full configuation is saved in a
   script named TEST1. Then, the example is re-run using the saved
   script.

GUI WITHOUT SCRIPT:

   At the command prompt enter:
      1) ui <enter>
         (The GUI begins executing in a window titled: ICE Version 5.5.)
         Enter the following menu commands:

      2) File ==> Load KB
         (Select rpl.flatfile from the "Select Knowledge Base" pop-up list
          and click "OK".)

      3) Click "OK" in the pop-up window notifying the user that 107
         objects have been loaded.

      4) Utilities ==> Open Toolbox ==> Reasoner
         (The Reasoner begins executing in its window.)

      5) Click the large rectangular box at the bottom of the window
         titled "Click Here to Connect"
         (The Reasoner window fills in its Monitor, Simulator, Fault Detector
         and Diagnoser panes.)

      6) In the upper left corner of the window, click "Download KB".
         (Several messages appear in the Fault Detector pane of the Reasoner
         window informing the user of tolerance settings that have been established.)

      7) Utilities ==> Open Toolbox ==> Data Provider
         (The Data Provider begins executing in its window.)

      8) Click the large rectangular box at the bottom of the window
         titled "Click Here to Connect".
         (The Data Provider window fills in its Data Source pane including tape
         "player" buttons and the status pane.

      9) Click the "New" box near the bottom of the Data Source window.

     10) Click "a1open.kpd" in the pop-up window that appears and click "OK".
         (The status pane fills in with the Start: and Stop: times recorded in
         the .kpd data file.)

     11) Click Configure ==> 3.
         (The window divides into three panes, one large pane on the left and two
         smaller panes on the right of the screen.)

     12) Click Utilities ==> Reasoner (to designate that the Reasoner should run in
         the large left-hand window (Pane 1)).
         (The Reasoner begins running in Pane 1.)

     13) Click Utilities ==> Data Provider (to designate that the Data Provider
         should run in the upper pane of the right-hand window (Pane 2)).
         (The Data Provider begins running in Pane 2.)

     14) Click the Monitor Status "On" radio button in the Reasoner
         pane on the left of the screen.

         It only possible to click the Monitor Status "On" radio button within the
         Reasoner after the Data Provider has been started. (There is nothing for
         the Reasoner to "monitor" until a data source has been specified.)
         Turning the Monitor On also instructs the Reasoner to connect to the Data
         Provider and vice versa.  The connection between the two processes is
         indicated by a bright green wavy arrow in the top of the Reasoner and
         Data Provider windows.)

         Note: The system can be run in "simulate only" mode with no data
         provider. In the simulate only mode, archiving can be turned on to
         generate an archive.dat file.  This file can be renamed "testXYZ.kpd"
```

or something similar and subsequent runs can be made using this kpd file in the Data Provider to test the full-up KATE system with simulated data. To run in the simulate only mode, do not click the Monitor status "On" radio button. Instead, click the Simulator Status radio button to "On", then click the Archiving radio button "On". Click the "Start" button in the upper left of the Reasoner pane and manipulate commands in the Tree Display Pane as described below.

15) Click the Simulator Status radio button to "On", the Fault Detector Status radio button to "On", the Diagnoser Status radio button to "On", and click the "Start" button in the upper left of the Reasoner pane.

16) Click the green "Play" button in the Data Provider window.

    (The Data Provider begins sending data to the reasoner. The GMT: time display in the middle of the upper pane of the KATE window begins updating with the time sent by the data provider. After about 50 seconds, the Fault Detector Messages appear informing the user that the measurements F1, P6 and P3 are out of tolerance. After 56 seconds these three measurements are announced as "discrepant" and an automatic diagnosis begins. Two messages appear in the Diagnoser messages pane of the Reasoner. The messages name the suspect component ANNIN1 valve as the cause of the discrepancies and posit two failed VALUES for ANNIN1 : 1.0 (full open) or 0.8 (80% open). Both of these values are consistant with the current state of the system measurements.)

17) After these messages appear, click "Stop" in the top of the Reasoner pane to stop KATE's execution.

18) Click File ==> Save UI Environment. Type a name for the UI Environment file in the File Box popup and click "OK". Any filename will do. In this example type TEST1 in the Selection window. (This action saves the UI Environment for use later.)

19) Click File ==> Exit and confirm with "OK" in the Verify Exit pop-up window.
    (The KATE program exits returning the user to the Linux Terminal window.)

The following example illustrates how to run the GUI with the TEST1 script that has just been created.

GUI WITH TEST1 SCRIPT FILE:

At the command prompt in the Linux Terminal window enter:

1) ui -uienv TEST1 <enter>

    (The GUI begins executing in a window titled: ICE Version 5.5.) The window has three panes. The large pane on the left is filled with the Reasoner. The small pane in the upper right is filled with the Data Provider and the small pane on the lower right is empty. Note that the KB (knowledge base) rpl.flatfile has already been loaded by the script, so loading a KB is not necessary.

Enter the following menu commands:

2) Click the large rectangular box at the bottom of the Reasoner window titled "Click Here to Connect".
   (The Reasoner window fills in its Monitor, Simulator, Fault Detector and Diagnoser panes.)

3) In the upper left corner of the window, click "Download KB".
   (Several messages appear in the Fault Detector pane of the Reasoner window informing the user of tolerance settings that have been established.)

4) Click the large rectangular box at the bottom of the Data Provider window titled "Click Here to Connect".
   (The Data Provider window fills in its Data Source pane including tape "player" buttons and the status pane.

5) Click the "New" box near the bottom of the Data Source window.

6) Click "a1open.kpd" in the pop-up window that appears and click "OK". (The status pane fills in with the Start: and Stop: times recorded in the .kpd data file.)

7) Click the Monitor Status "On" radio button in the Reasoner pane on the left of the screen.

   (The connection between Data Provider and Reasoner processes is indicated by a bright green wavy arrow in the top of the Reasoner and Data Provider windows.)

8) Click the Simulator Status radio button to "On", the Fault Detector Status radio button to "On", the Diagnoser Status radio button to "On", and click the "Start" button in the upper left of the Reasoner pane.

9) Click the green "play" button in the Data Provider window.

   (The Data Provider begins sending data to the reasoner, the Fault Detector and Diagnoser Messages appear as before.)

10) After these messages appear, click "Stop" in the top of the Reasoner pane to stop KATE's execution.

Running KATE with text ui

The reasoner may be run without the GUI by using the rm-ui program to control the reasoner. The reasoner and the rm-ui can be run in either quiet, brief or verbose mode; each level shows more detail on the messages being sent between the two processes. Typing "m" for any text ui will provide a menu of options.

```
reasoner [ -q|b|v ] [ -c channel ]

rm-ui [ -q|b|v ] [ -c channel ]
```

A socket or channel is set up between processes in the KATE system the default channel number is 1. If a reasoning processes is already running the user can specify another channel number to be used.

The reasoner program is started first and then the rm-ui program.

The Data Module process can also be run with its own text user interface.

```
Ethernet data provider
edp [ -q|b|v ] [ -c channel ]
eui [-h hostname] [-c channel] [-s source] [-bf buildfile] [-q|b|v ]
    [-run]
```

The source is the firing room number (sort of), the run option tells the text ui to start the edp program automatically.

Note: This functionality is not available in the latest version.

```
Playback data provider
pdp [ -c channel ] [ -l tapefile] [ -q|b|v ]
pui [ -h hostname ] [ -c channel ] [ -q|b|v ]
```

Other programs:

1) build [-t tcidname] [-bf buildname] [-q|b|v]
   The tcidname should be just the name of the TCID without any extension (e.g., sa065a), the buildname is a name of a file that contains a list of fds (one on each line) in the system that the application needs. A valid usage might look like:

   ```
   build -t sa065a -bf fds.nam -v
   ```

```
           The -v turns on the debugging messages.  The above command line will
           generate a build file called fds.bld.

        2) translator [-i inputfile] [-o outputfile] [-a]
           The translator program takes a KATE database file and outputs a KATE-readable
           flatfile. See KATE/src/translator/translator.c for required database format.

Other files

        The file fds.nam contains list of fds used as input to the build
        program.

        TCID files *.bsd *.bsi also used as input to the build program
        Note that the build program has not been upgraded to work with PCGOAL2.
```

<u>Knowledge base instructions</u>

The mini-model (see Figure A-1) was constructed as a group of KATE-C objects. The README.flatfile text included with the software describes how to create a KATE model or knowledge base (KB). As explained in the installation instructions above, the KB is made up of two parts:

a) A library of components that are compiled into the C++ system, each library object has its input names and output function (a single output function for each component is used). The class libraries are broken up into application-specific files and located in KATE/src/kb/<application>, where <application> is fcl, lox, alo, etc. All application source libraries are compiled into one complete class library - so there can be no duplication of class names between applications. The Rapid Propellant loading class libraries were included in the KATE/src/kb/alo directory.

b) A "flatfile" textual description of the model or KB specifying library components and how things are connected together as well as parameter values. See KATE/README.flatfile for a brief knowledge base generation guide.

c) The Rapid propellant loading KB is included in a file in the KATE/sun/exe/ directory called rpl.db. rpl.db describes each of the components in the mini-model and how they are connected to the others. Parameters such as tank sizes, tolerances for measurements, flow coefficients, equipment locations, minimum and maximum values are also included and documented in rpl.db

A8. HYDE-SPECIFIC ITEMS

<u>HyDE Software</u>
The HyDE software distribution is included with this report in a WinZip repository named HyDE.zip. The following directories are in the distribution:

- **doc** contains all the help files, manuals and tutorials describing how to build and save models, how to build the HyDE tool, how to configure HyDE for diagnosis, and how to run the HyDE tool.
- **ModelingParadigm** contains the HyDE modeling paradigm as well as three subdirectories containing model interpreters:
    - HyDEInterpreter contains the C++ code that allows user to check and save models, create, edit and save harness, and run scenarios.
    - HyDEGenericDecorator contains the C++ code that decorates the icons for model entities based on their attributes.
    - lib contains the HyDE, HyDEInterpreter, and HyDEGenericDecorator libraries.
- **models** contains all the HyDE models developed using the Generic Modeling Environment (GME) tool from Vanderbilt University.
- **src** contains all the C++ code that is part of the HyDE Reasoning Engine.
- **test** contains unit tests, system tests and regression tests.

<u>HyDE Rapid Propellant Loading models</u>
Hyde model source code for Rapid Propellant Loading with comments is included in the WinZip repository named HyDE_AFRL_Models.zip.

The changes that were necessary to configure HyDE for Rapid Propellant Loading are described in the PowerPoint file named AFRL_Demo-v0.pptx that is included in the HyDE directory with this report.

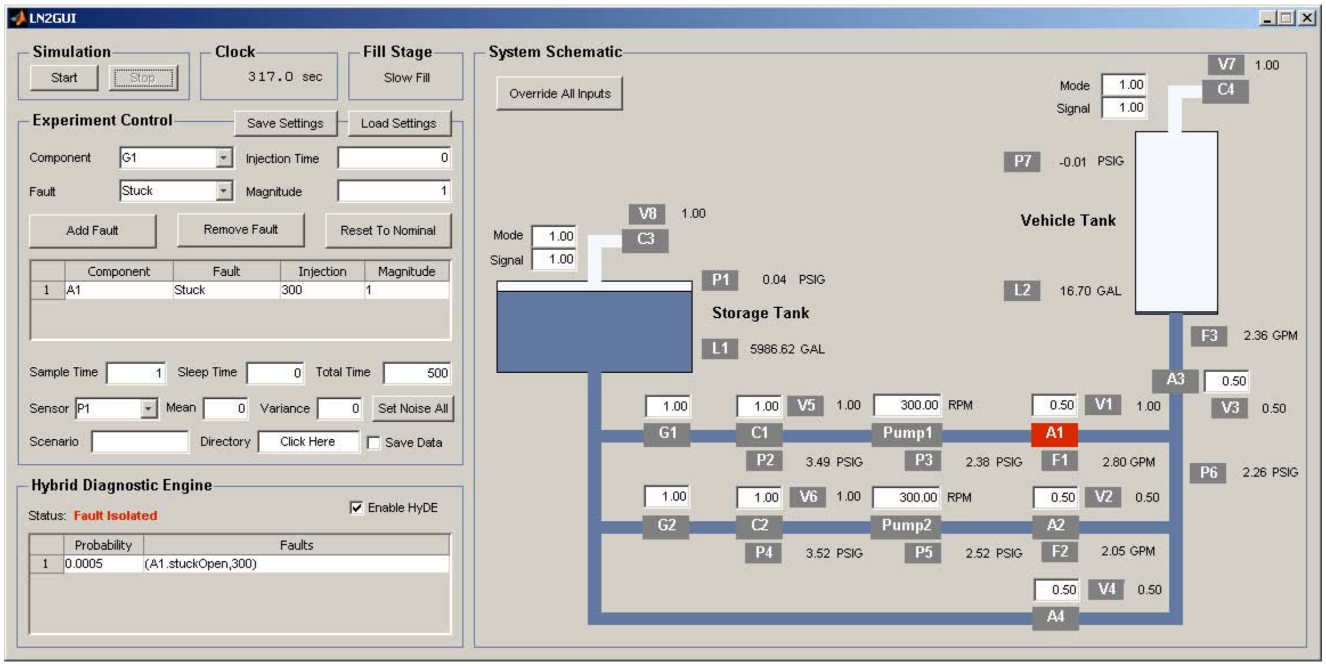
A9. SIMULINK/HYDE ITEMS

Rapid Propellant Loading HyDE movies

A set of HyDE demonstrations is included with this report as ".avi" movie files. They may be played with any movie player that supports the .avi format, such as Windows Media Player. The movies were created using the RPL simulation software. Three scenarios were selected to highlight the capabilities of HyDE. The movies are included in the **Movies** directory with the software accompanying this report.

Demonstration 1: Flow Control Valve Stuck Open

In the first scenario, a flow control valve, A1, becomes stuck open during the slow fill stage. This scenario is demonstrated in the movie file "HyDE-A1StuckOpen.avi". The scenario is illustrated in the screenshot below.



Clock: 317.0 sec

Fill Stage: Slow Fill

Experiment Control

Component: G1 | Injection Time: 0

Fault: Stuck | Magnitude: 1

	Component	Fault	Injection	Magnitude
1	A1	Stuck	300	1

Sample Time: 1 | Sleep Time: 0 | Total Time: 500

Sensor: P1 | Mean: 0 | Variance: 0

Scenario: | Directory: | ☐ Save Data

Hybrid Diagnostic Engine

Status: Fault Isolated | ☑ Enable HyDE

	Probability	Faults
1	0.0005	(A1.stuckOpen,300)

V7 1.00; Mode 1.00; Signal 1.00; P7 -0.01 PSIG; L2 16.70 GAL; F3 2.36 GPM; A3 0.50; V3 0.50; P6 2.26 PSIG

V8 1.00; Mode 1.00; Signal 1.00; P1 0.04 PSIG; L1 5986.62 GAL

G1 1.00; C1 1.00; V5 1.00; Pump1 300.00 RPM; A1 0.50; V1 1.00; P2 3.49 PSIG; P3 2.38 PSIG; F1 2.80 GPM

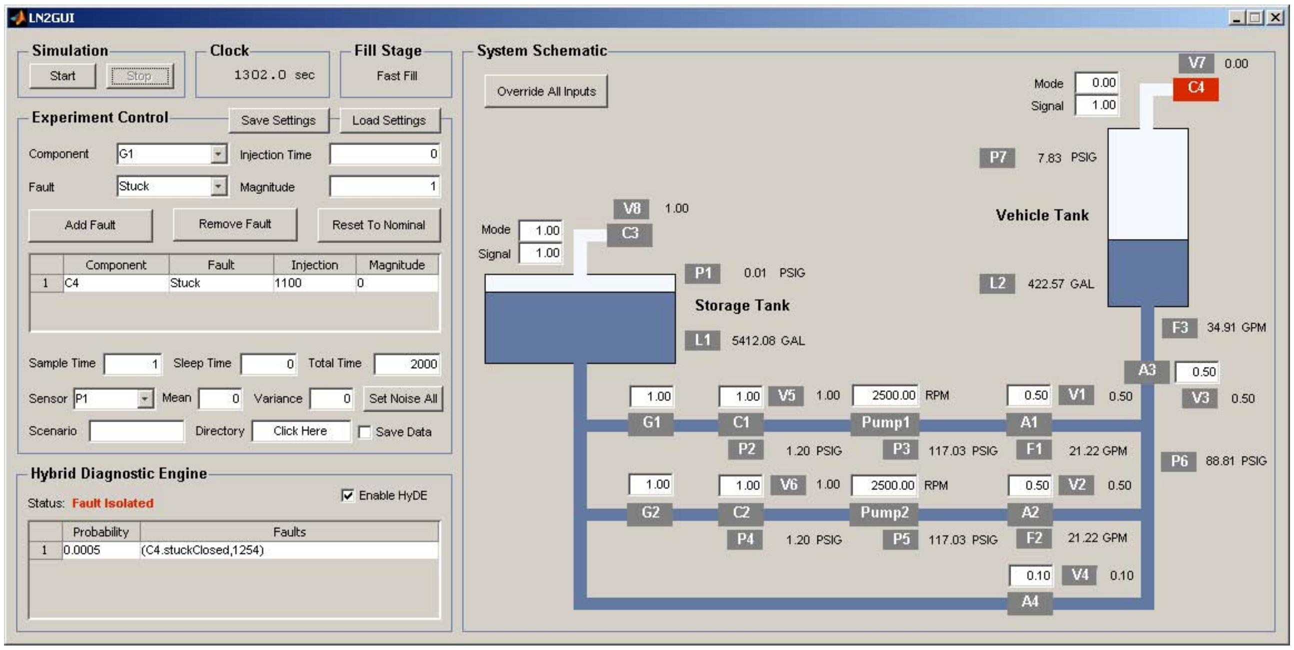
G2 1.00; C2 1.00; V6 1.00; Pump2 300.00 RPM; A2 0.50; V2 0.50; P4 3.52 PSIG; P5 2.52 PSIG; F2 2.05 GPM

A4 0.50; V4 0.50

Figure A-3
Flow Control Valve Stuck Open Scenario

It is important to first note that HyDE does not use the valve sensors, only flow, pressure, and level sensors are used for diagnosis. The system begins in the initialization phase, and begins slow fill at 100 seconds. The movie shows the output of flow sensor F1, which measures the flow through the first transfer line. An increase in the flow is observed at this time due to the change in control inputs. At 300 seconds, the fault is injected. A sharp increase in F1 is visible (the dotted black line in the plot indicates the nominal value at this point, and the solid blue line indicates the sensor output). HyDE immediately detects and isolates the fault at this time. At around 330 seconds, the operator reconfigures to resume a nominal flow rate through the transfer line by reducing the Pump 1 input to 275 RPM.

Demonstration 2: Vehicle Tank Vent Valve Stuck Open
In the second scenario, the vehicle tank vent valve, C4, becomes stuck closed during the slow fill stage. This scenario is demonstrated in the movie file "HyDE-C4StuckClosed.avi". The scenario is illustrated in the screenshot below.



Clock: 1302.0 sec
Fill Stage: Fast Fill

Experiment Control

Component: G1 Injection Time: 0
Fault: Stuck Magnitude: 1

	Component	Fault	Injection	Magnitude
1	C4	Stuck	1100	0

Sample Time: 1 Sleep Time: 0 Total Time: 2000
Sensor: P1 Mean: 0 Variance: 0
Scenario: Directory: Click Here ☐ Save Data

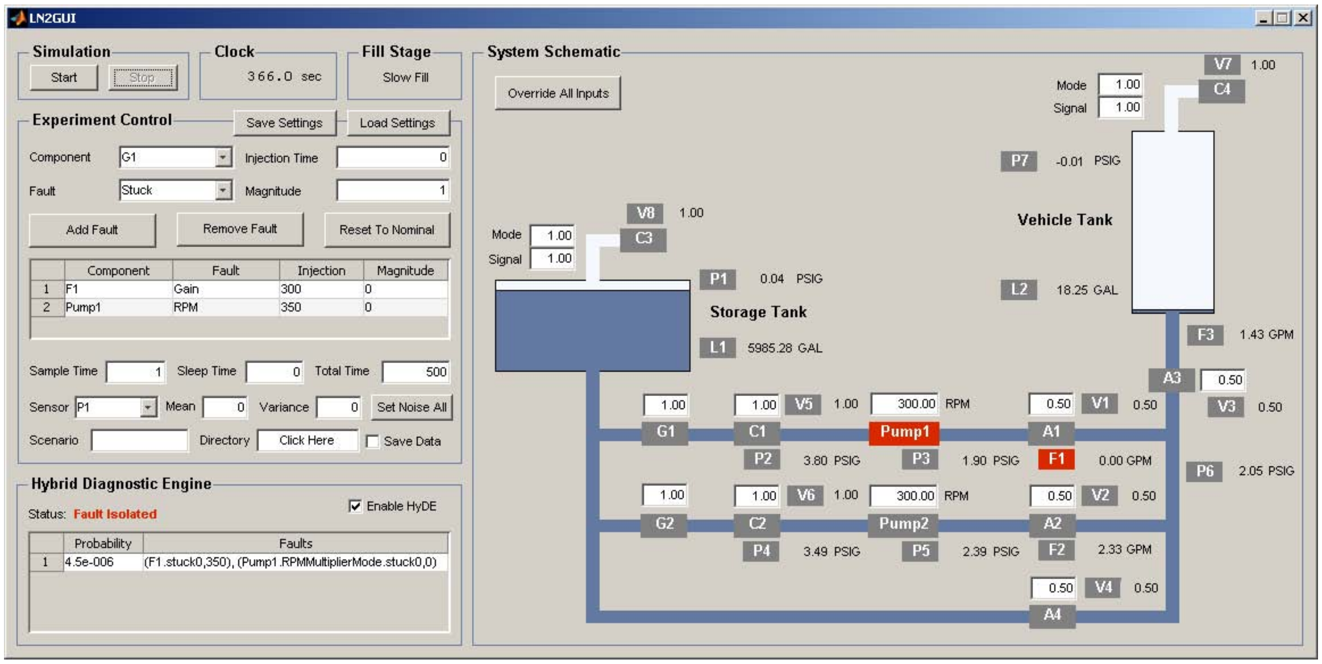
Hybrid Diagnostic Engine
Status: Fault Isolated ☑ Enable HyDE

	Probability	Faults
1	0.0005	(C4.stuckClosed,1254)

Figure A-4
Vehicle Tank Vent Valve Stuck Open Scenario

The system begins in the initialization stage and proceeds to slow fill at 100 seconds, and fast fill at 600 seconds. As the vehicle tank fills, the ullage pressure, measured by P7, begins to increase. The movie shows the output of P7. In nominal operation, the vent valve C4 automatically opens when the ullage pressure reaches 7 psig, and closes when the ullage pressure reaches 3 psig. In this way, it maintains the ullage pressure within safe levels. If the vent valve fails to open at the required time, the ullage pressure may build up to unacceptable levels and an abort will have to be issued. Therefore, it is important to quickly identify such failures. The vent valve C4 becomes stuck closed at 1100 seconds. At this time, it is supposed to remain closed, as the ullage pressure is not yet at 7 psig. At 1254 seconds, the ullage pressure begins to exceed 7 psig. HyDE immediately recognizes that C4 has failed. Around 1340 seconds, the operator initiates drain back to safe the system. The pumps are turned off and all valves are fully opened.

Demonstration 3: Flow Sensor Failure and Pump Failure
In the third scenario, two faults occur. The flow sensor F1 fails, followed by a failure of Pump 1 during the slow fill stage. This scenario is demonstrated in the movie file "HyDE-F1FailPump1Fail.avi". The scenario is illustrated in the screenshot below.



Clock: 366.0 sec
Fill Stage: Slow Fill

Component: G1 Injection Time: 0
Fault: Stuck Magnitude: 1

	Component	Fault	Injection	Magnitude
1	F1	Gain	300	0
2	Pump1	RPM	350	0

Sample Time: 1 Sleep Time: 0 Total Time: 500
Sensor: P1 Mean: 0 Variance: 0
Scenario: Directory: Click Here ☐ Save Data

Status: Fault Isolated ☑ Enable HyDE

	Probability	Faults
1	4.5e-006	(F1.stuck0,350), (Pump1.RPMMultiplierMode.stuck0,0)

Mode 1.00 Signal 1.00 (C4); V7 1.00
P7 -0.01 PSIG
Mode 1.00 Signal 1.00 (C3); V8 1.00
P1 0.04 PSIG
L2 18.25 GAL
L1 5985.28 GAL
F3 1.43 GPM
A3 0.50
V3 0.50
G1 1.00; C1 1.00; V5 1.00; Pump1 300.00 RPM; A1 0.50; V1 0.50
P2 3.80 PSIG; P3 1.90 PSIG; F1 0.00 GPM; P6 2.05 PSIG
G2 1.00; C2 1.00; V6 1.00; Pump2 300.00 RPM; A2 0.50; V2 0.50
P4 3.49 PSIG; P5 2.39 PSIG; F2 2.33 GPM
A4 0.50; V4 0.50

Figure A-5
Flow Sensor Failure and Pump Failure Scenario

The system begins in the initialization stage, and begins slow fill at 100 seconds. The movie shows the output of flow sensor F1. At 300 seconds, the sensor fails and returns only a value of zero. HyDE immediately determines that the flow sensor has failed in this way. At this point, only the remaining flow sensors, F2 and F3, are available for diagnosis. The movie then shows the output of F3 instead of F1. At 350 seconds, Pump 1 fails. HyDE immediately detects that something else has gone wrong and initially pinpoints a failure of A1 (this is not visible in the movie). At the next time step, HyDE corrects its diagnosis and pinpoints Pump 1 as the second failure. (Note that if the pressure and level sensors are also used, HyDE will immediately diagnose Pump 1 as the second failure, and not A1.) At around 380 seconds, the operator reconfigures the system to regain nominal flow to the vehicle tank despite the failure of Pump 1. Valve A1 is closed, valve A2 is fully opened, and the RPM of Pump 2 is increased to 375. The flow to the vehicle tank is then restored to the nominal value of about 2.2 GPM.

A10. RAPID PROPELLANT LOADING HYDE / SIMULINK DEMONSTRATION

A set of HyDE software demonstrations is included with this report as Matlab Simulink files. The files are located in the **Matlab Simulink files** directory with the software accompanying this report. They may be executed in the Matlab environment. The following instructions describe how to run the software:

LN2 Matlab/Simulink Simulation

This section describes the Matlab/Simulink Simulation of the simplified LN2 test bed. The scope of the model is shown below.

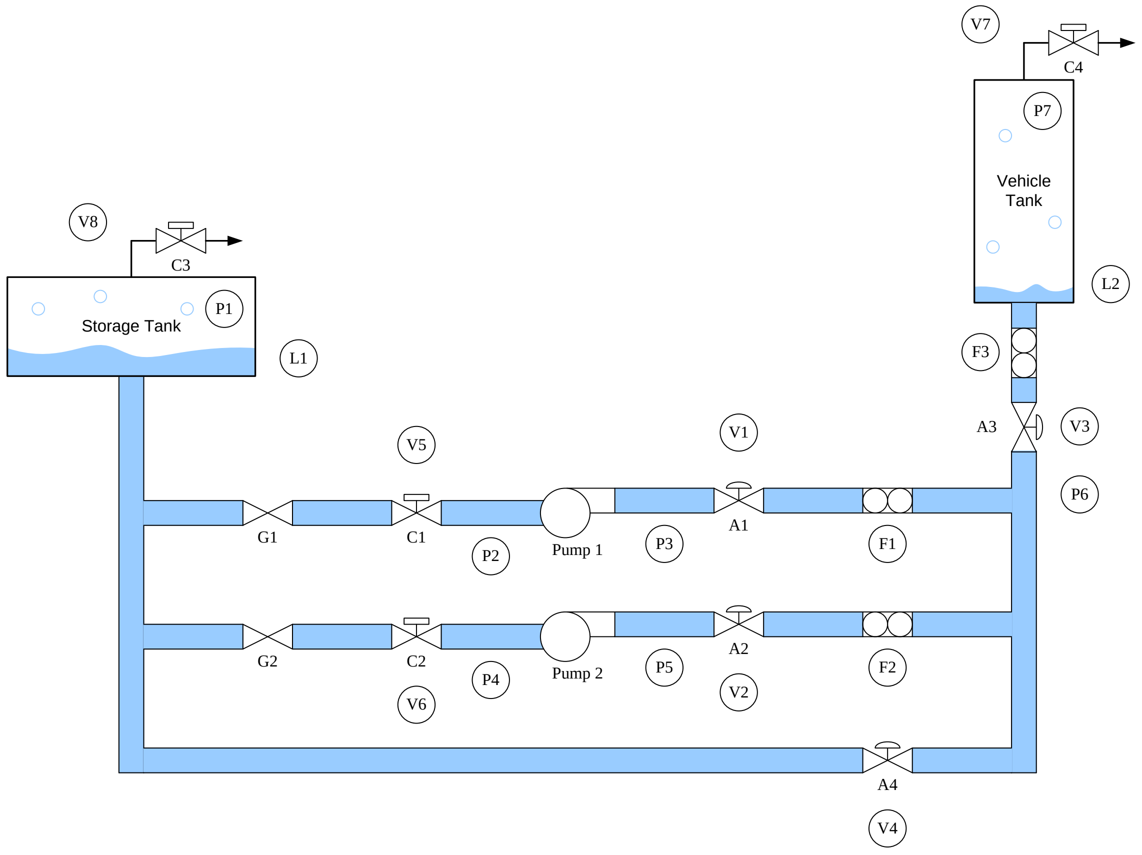


Figure A-6
LN2 Test Bed Simulation

A large storage tank holds the propellant. A pump is used to create a pressure differential between the two tanks in order to move fluid from the storage tank to the vehicle tank. Flow (F), pressure (P), level (L), and valve position (V) sensors are included, as denoted in the above schematic. Note that this simulation represents a simplified propellant loading model. In particular, we neglect boiling/condensation processes, heat flow, and temperatures (a constant gas temperature is assumed).

Simulation Overview

The simulation consists of a Simulink model, an associated graphical user interface (GUI), and a set of Matlab scripts and function, which should all be kept in the same directory. The main files are:

- *LN2Sim.mdl*: The Simulink model of the simplified LN2 test bed.
- *LN2Params.m*: A Matlab script that sets the values of model and configuration parameters.
- *LN2GUI.fig*: The simulation GUI.
- *LN2Control.m*: A Matlab function that implements the automatic filling protocol.

The top-level of LN2Sim.mdl is shown below.

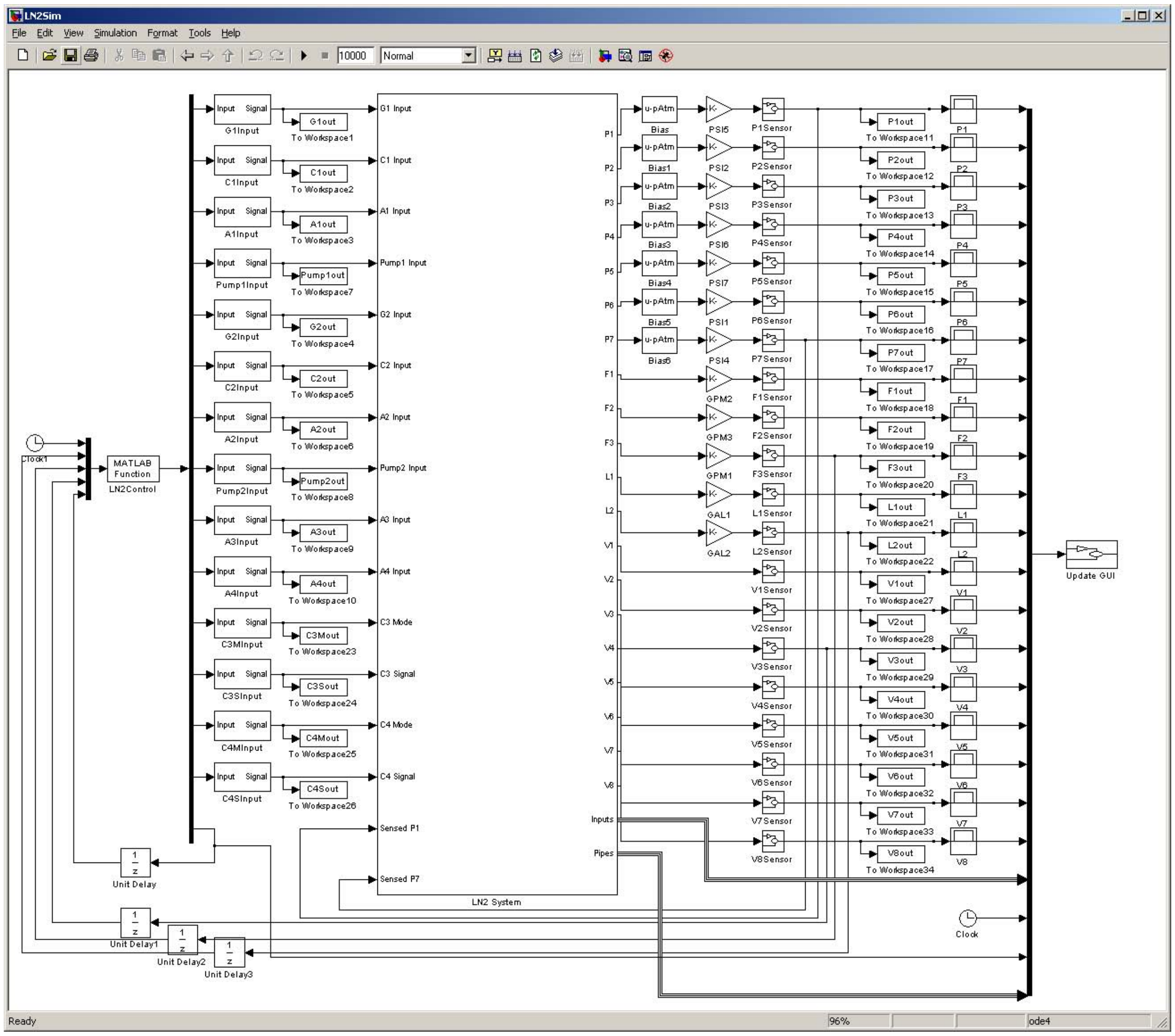


Figure A-7
Top Level LN2 Simulation Model

The model mainly consists of the control block, which calls LN2Control.m, input blocks, the main system block (which implements all the physical processes of the simulation), sensor blocks, and output blocks. Outputs are fed back into the control block for closed-loop control.

The values of the inputs are determined by the current filling stage, specified by a set of parameters indicating when the various stages begin. The ‘LN2Control’ block, which calls LN2Control.m, determines the stage. The set of stages are:

- *Initialization*: All manual and discretely-controlled valves are set open, and all flow control valves are set at 50%. The pumps are turned off.
- *Slow Fill:* The pumps are set to 300 RPM. Slow fill begins when the simulation time is ‘tSlow’.
- *A4 Test*: Valve A4 is set to 10%. This occurs when the simulation time is ‘tA4’, which should be greater than ‘tSlow’.
- *Fast Fill*: The pumps are set to 2500 RPM. Vent valve C4 is placed into autonomous mode, where it will open at 7 psig and close at 2 psig. Fast fill begins when the simulation time is ‘tFast’, which should be greater than ‘tA4’.
- *Topping*: Pump speeds are lowered, A4 is partially opened more than 10%, and A3 is controlled to maintain F3 at 20 GPM. Topping occurs when the tank level reaches 980 gallons, which should occur after ‘tFast’.
- *Replenish*: A4 is opened to a greater extent. Vent valve C4 is set open. Replenish begins when the tank level reaches 1000 gallons.
- *Drainback*: All flow control valves are set to 100%. The pumps are turned off. Drainback occurs when the simulation time is ‘tDrain’, which should be after L2 reaches 1000 gallons.
- *End*: Vent valve C4 is closed. This occurs when L2 reaches 10 gallons.

Component Models

In this section, we describe the models of the system components and their faults. The simulation model is constructed in a component-based fashion using a component library. In this framework, a system model is created by instantiating different components and connecting them appropriately. In this way, different system configurations can be easily developed. LN2Library.mdl contains the component library. It is shown below.

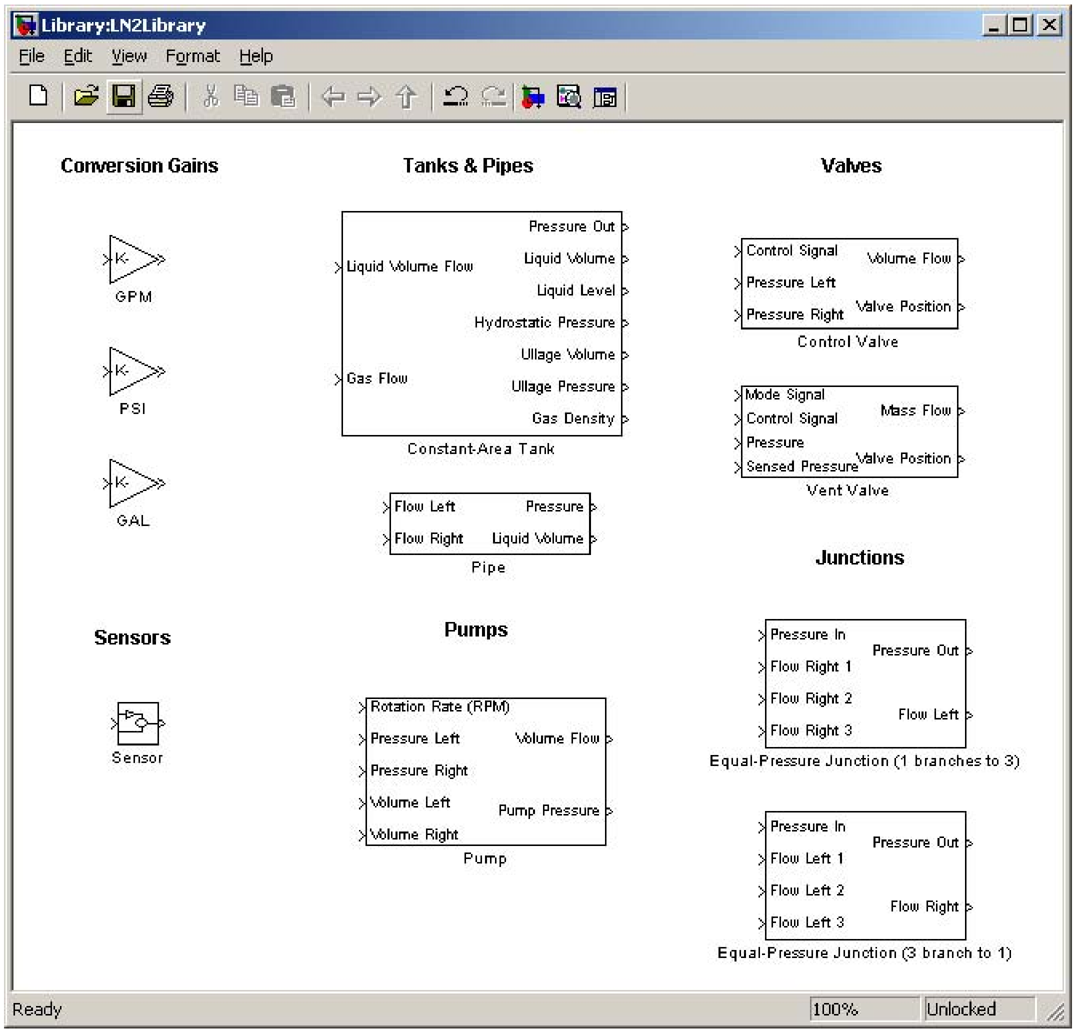
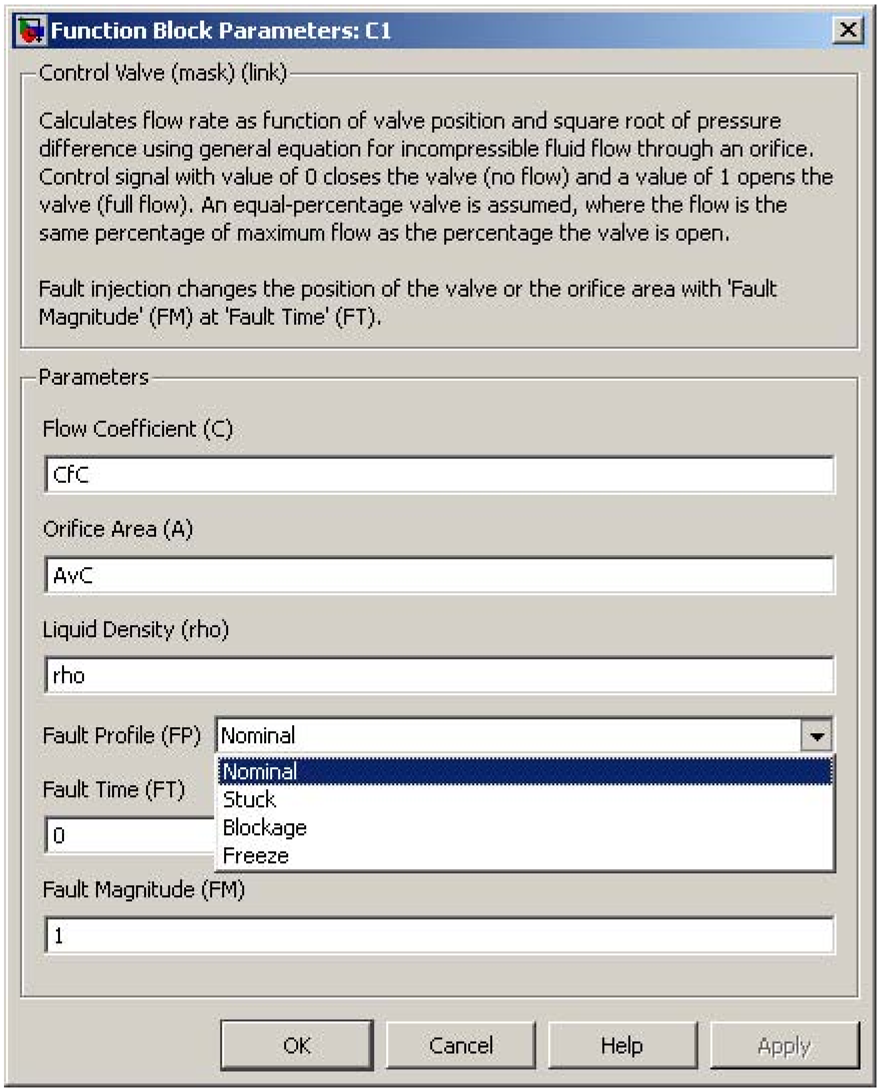


Figure A-8
LN2 Component Library

The system has been divided into tanks, pipes, control valves, vent valves, pumps, junctions, and sensors. Each component model is *masked*, allowing component parameters to be set through a dialog. Double-clicking on a component model brings up the component dialog. A simple description of the component is provided in the mask dialog. Expanded descriptions are available by clicking on the help button on the mask dialog.

Note that the variable names in parentheses shown in the mask prompt are the corresponding internal variables used by the mask. For example, liquid density is given the variable name of ‘rho’ in many component models. Internal block computations refer to the user-entered value for liquid density using the variable name ‘rho’.

As an example, the mask prompt for the C1 control valve is shown below.



Function Block Parameters: C1

Control Valve (mask) (link)

Calculates flow rate as function of valve position and square root of pressure difference using general equation for incompressible fluid flow through an orifice. Control signal with value of 0 closes the valve (no flow) and a value of 1 opens the valve (full flow). An equal-percentage valve is assumed, where the flow is the same percentage of maximum flow as the percentage the valve is open.

Fault injection changes the position of the valve or the orifice area with 'Fault Magnitude' (FM) at 'Fault Time' (FT).

Parameters

Flow Coefficient (C): CfC

Orifice Area (A): AvC

Liquid Density (rho): rho

Fault Profile (FP): Nominal

Fault Time (FT): 0

Fault Magnitude (FM): 1

Figure A-9 - Sample Mask Prompt

Descriptions of the components are as follows.

- **Tank**

  Computes amount of liquid and gas mass and corresponding pressures. Two mass balance equations and corresponding state variables are included, one for the liquid, and one for the gas. State changes between liquid and gas are not modeled and a constant gas temperature is assumed.

  Given the liquid mass, the height of the liquid is computed based on the known liquid density (rho) and the area of the tank (A). Hydrostatic pressure is computed using the height of the liquid plus the elevation of the tank using $\rho g(h+e)$, where ρ is the liquid density, g is the acceleration due to gravity, h is the liquid height, and e is the elevation. Given the gas mass, ullage pressure is added to that total, calculated using the ideal gas law assuming a known gas temperature.

- **Pipe**

  Models a pipe as a capacitance. Includes liquid volume in the pipe as a state variable. Integrates the sum of flows to obtain volume of liquid in the tank, given the initial volume of liquid (I). Dividing volume by capacitance (C) provides the pressure:

  $$p = I + \frac{1}{C}\int \Delta f \qquad \text{Equation A-1}$$

  where Δf is the sum of flows (flow from left – flow from right).

- **Control Valve**

  Calculates flow rate as function of valve position and square root of pressure difference. Control signal with value of 0 closes the valve (no flow) and a value of 1 opens the valve (full flow). A linear flow valve is assumed, where the flow is the linear with the percentage the valve is open.

  The flow equation is based on flow through an orifice with area A and flow coefficient C, where the volume flow rate is given by:

  $$C \cdot A\sqrt{\frac{2}{\rho}|\Delta p|} \cdot sign(\Delta p) \qquad \text{Equation A-2}$$

  Where C is the flow coefficient, A is the orifice area, Δp is the pressure drop and sign is the direction of flow. Fault injection changes the position of the valve or the orifice area with 'Fault Magnitude' (FM) at 'Fault Time' (FT). Profiles include:

  - 'Nominal', where all nominal parameters are used
  - 'Stuck', where the degree open is FM in [0,1]
  - 'Blockage', where actual orifice area = nominal orifice area * FM
  - 'Freeze', where the degree open is permanently set at the value one time step (determined by sampleTime) before the fault

- **Vent Valve**

  Models simplified nonchoked flow of gas through an orifice with area A and discharge coefficient C. Mode signal of 1 indicates external control/override, 0 indicates autonomous. When in controlled mode, the external control signal determines the valve position. When in autonomous mode, the valve opens when pressure reaches opening pressure (openP) and closes when pressure reaches closing pressure (closeP).

  The gas flow equation used is given by:

  $$C \cdot A\sqrt{\left|(p - pAtm)\right|} \cdot sign(p - pAtm), \qquad \text{Equation 3}$$

  where *C* is a flow coefficient, *A* is the orifice area, *p* is the pressure at the valve (e.g., ullage pressure), and *pAtm* is the atmospheric pressure.

  Fault injection changes the position of the valve or the orifice area with 'Fault Magnitude' (FM) at 'Fault Time' (FT). Profiles include:

  - 'Nominal', where all nominal parameters are used
  - 'Stuck', where the degree open is FM in [0,1]
  - 'Blockage', where actual orifice area = nominal orifice area * FM
  - 'Freeze', where the degree open is permanently set at the value before the fault

- **Pump**

  Computes pump pressure and flow through the pump given RPM and pump suction and discharge pressures as input.

  The pump pressure is computed as:

  $$p_{pump} = \frac{p_{\max} \cdot \omega^2}{\omega_{\max}^2}, \qquad \text{Equation 4}$$

  where p_{max} is the maximum pump pressure (in psi), ω is the current pump rotation rate (in RPM, given as input), and ω_{max} is the rotation at p_{max}. The pump pressure as computed by this equation is then converted to Pascals for internal model computations. The pressure difference is then computed, and the flow is computed using the general equation for incompressible fluid flow through an orifice, given above.

  Fault injection allows changes in the RPM and blockage specified by 'Fault Magnitude' (FM) and 'Fault Time' (FT). Profiles include:

  - 'Nominal', where all nominal parameters are used
  - 'RPM’, where actual RPM = input RPM * FM
  - 'Blockage', where actual CA = nominal CA * FM

- **Junction**

  Junctions are not masked. The library contains only equal-pressure junctions. The input pressure determines the output pressure. The input flows are summed to compute the output flows to ensure that the net flow through the junction is zero.

- **Sensor**

  Generic sensor model. Includes additive Gaussian noise specified by 'Noise Mean' and 'Noise Variance' to the input value to produce the output value. Specify mean and variance as zero to exclude noise.

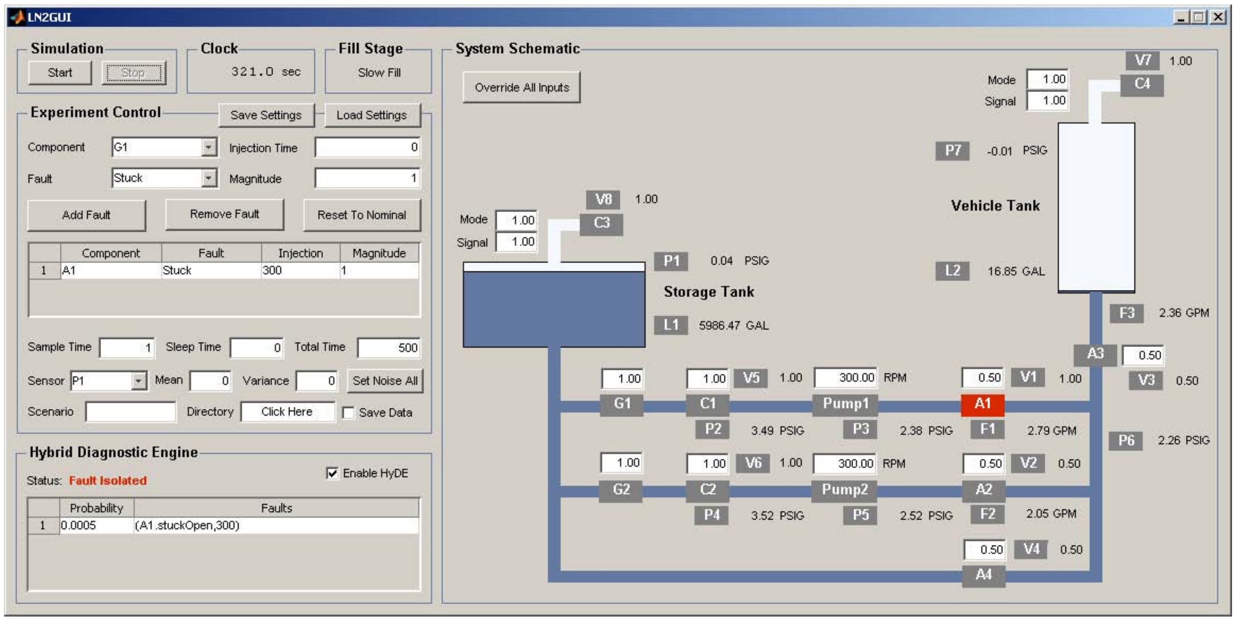
  Fault injection modifies the input value to one of several profiles (FP), starting at 'Fault Time' (FT) with magnitude specified by 'Fault Magnitude' (FM). Profiles include:

  - 'Nominal', where output=input+noise
  - 'Gain', where output=FM*input+noise
  - 'Drift', where output=input+FM*(FT-t)+noise
  - 'Bias', where output=input+FM+noise
  - 'Stuck', where output=FM+noise
  - 'Freeze', where output=o+noise, and “o” is permanently set at the value one time step (determined by sampleTime) before the fault
  - 'Noise', where output=input+FM*noise

Running the Simulation

To run the simulation, first open LN2Sim.mdl. The GUI will open automatically, and close automatically when the simulation is closed. When closed, the GUI will reset the simulation to the nominal scenario unless the simulation is already closed. If the GUI is accidentally closed, run the LN2GUI.m function in the command prompt to reopen it. The simulation can be used without the GUI. To do so, set useGUI=0 in the command prompt. Alternatively, set useGUI=0 in LN2Params.m to make that the default setting.

A screenshot of the GUI is shown below.


LN2GUI

Clock: 321.0 sec

Fill Stage: Slow Fill

Experiment Control

Component: G1 | Injection Time: 0

Fault: Stuck | Magnitude: 1

	Component	Fault	Injection	Magnitude
1	A1	Stuck	300	1

Sample Time: 1 | Sleep Time: 0 | Total Time: 500

Sensor: P1 | Mean: 0 | Variance: 0

Scenario: | Directory: Click Here | ☐ Save Data

Hybrid Diagnostic Engine

Status: Fault Isolated | ✓ Enable HyDE

	Probability	Faults
1	0.0005	(A1.stuckOpen,300)

Figure A-10 - LN2 Simulation Graphic User Interface

The GUI has the following functionality.

- *Simulation control:* The simulation can be started, stopped, and paused from the GUI. The GUI will display the current clock time, current fill stage, all measured values, and all current input values being supplied by the control.

- *Input/Output display:* The current inputs being supplied to the system and the current set of sensor values are displayed on the GUI. To manually control the system, enter a new value in the input's text box. The text will become bold to indicate that it is being overridden. Inputs of arbitrary precision can be provided, but it will only be shown up to two decimal places. When the simulation is started again, individual overrides are turned off for all inputs. To override all inputs, use the "Override All Inputs" button. This will put the system into a new fill stage controlled only by the user. All inputs will remain as they were at the time the button was pressed unless changed by the user. When manual override is turned off, the system will resume the nominal filling sequence and all inputs will return to their values for that fill stage, unless previously overridden individually. The GUI will also plot the data histories for the inputs and sensors. When the simulation is stopped or paused, the current data history for an input or sensor is plotted by clicking on the gray box with the input or sensor name.

The valve inputs are specified by a number in the interval [0,1], representing the amount the valve is open, where 0 refers to fully closed (no flow), and 1 refers to fully open (full flow). The vent valves (C3 and C4) may be operated in an automatic mode, in which they open and close depending on ullage pressure. A mode input of 0 refers to the automatic mode, in which the signal input is ignored. When the mode input is 1, the signal input determines the position of the valve. For the pumps, the RPM value is specified. Each input is logged to the workspace.

- *Experiment Control:* The experiment control panel allows the user to inject faults, set sensor noise, specify a scenario name for saving data, and set timing parameters.

    - *Timing parameters:* All timing parameters are specified in seconds. The sample time parameter determines at what interval the GUI is updated, data is logged, and inputs are supplied to the system. The sleep time parameter specifies how long the GUI should pause the simulation between updates, in order to slow the simulation to real-time or some factor faster or slower than real –time. For no pausing, enter 0. For real-time synchronization, enter the current value of sample time. The total time parameter sets how long the simulation will run (in simulation time, not real time). The user can still stop the simulation early using the Stop button.

    - *Saving data:* Data is saved from the simulation if the scenario name field is nonempty and the Save Data checkbox is checked. The data will be saved to a .mat file named by the provided scenario name, in the directory specified in the directory box. To enter a directory, click on the gray directory box and a folder selection dialog will appear, and select the desired directory. HyDE and KATE files will be written with the same filename, only appended with “HyDE” and “KATE”, respectively, and written in the same directory. If no directory is specified the files will be saved to the current working directory. Saved data will be stored in a variable called ‘LN2Data’. When a .mat file is loaded, this is the variable name that Matlab will assign.

    - *Fault injection:* To inject a fault, first select the component from the drop-down menu. The faults drop-down menu will populate with the list of fault modes for that component (see the previous section). Set the injection time and magnitude in the corresponding boxes. Then push the “Add Fault” button. The fault will be added to the table below and injected into the simulation model. Only one fault mode per component is allowed. To remove a fault, select it from the table and push the “Remove Fault” button. To remove all faults, push the “Reset to Nominal” button.

    - *Sensor noise:* The sensor noise is additive white Gaussian noise, and the mean and variance can be specified. To set the noise of a particular sensor, select it from the drop-down menu. Enter new values for mean and/or variance. The simulation model will be updated with the new values. To set the noise of all sensors to the same mean and variance, enter the desired values in the mean and variance text boxes and push the “Set Noise All” button.

    - *Saving/Loading Settings:* As well as saving simulation data from a scenario, the settings of the scenario can also be saved in order to repeat the experiment at a later time. To save settings, push the “Save Settings” button. All the currently injected faults, current values of sensor noise, and current values of sample time, sleep time, and total time will be saved. A file dialog will appear. Enter the filename as a .mat file to save the settings. To load previously saved settings, push the “Load Settings” button. A file dialog will appear. Select the previously saved .mat file to load the settings. An error message box will appear if the file is not in the correct format (i.e., it was not generated by the “Save Settings” button.

Four settings files are already available for use: “NominalSettings.mat”, which loads settings for a nominal loading scenario, “A1StuckOpenSettings.mat”, which loads settings for a stuck-open fault in valve A1, “C4StuckClosedSettings.mat”, which loads settings for a stuck-closed fault in valve C4, and “F1FailPump1Fail.mat”, which loads settings for a F1 sensor failure followed by a Pump1 failure.

- *Hybrid Diagnostic Engine:* Results from the Hybrid Diagnostic Engine (HyDE) are displayed on the GUI in tabular format. To enable HyDE, check the “Enable HyDE” checkbox. If the HyDE executable (HyDEInterface.mexw32) cannot be found, the GUI will not allow the box to be checked. When HyDE detects a fault, the status indicator will display “Fault Detected”. When a fault is isolated, the status indicator will display “Fault Isolated” and the table will populate with the current set of diagnoser results. For each candidate, a probability and a set of faults with the estimated times of occurrence will be displayed. For the most probable candidate, the related components will be automatically highlighted in the schematic. For example, in the GUI shown above, the A1 component is highlighted. Selecting a specific row in the table will also highlight the components associated with that candidate.

Command Line Interface

In addition to the GUI, there are a set of functions available to access the simulation from the command line to facilitate running different scenarios and collecting data automatically. The simulation model must be open for functions that modify or execute the model to succeed. Many of these functions are also used by the GUI, and specific information can be found be running “help function” in the Matlab command prompt, where “function” is the name of the function for which information is needed. Brief summaries are given below.

Most of these functions rely on LN2System.mat, which is generated from LN2System.xls. The spreadsheet contains the names of all components, inputs, and outputs, descriptions of each, and the corresponding names in HyDE and KATE. If changes are made to LN2System.xls, LN2System.mat can be regenerated using “importSystemFile(‘LN2System.xls’);” at the Matlab command prompt.
The scripts and functions are as follows. Functions or scripts in bold are those which are top-level functions that are likely to be called directly. Functions which are not in bold are helper functions called by the top-level functions and probably do not need to be called directly.

- **sim(model)**

  This is a native Matlab command to run an open Simulink model. Use “sim(‘LN2Sim’);”

to simulate from the Matlab command line.

- **LN2Data = saveData(scenario,dir)**

  This function saves the outputs from LN2Sim to a .mat file into a variable named ‘LN2Data’, which it also returns as output. The “scenario” argument is a string that specifies the file name, and “dir” specifies the directory.

- **writeDataHyDE(filename,data)**

  This function writes the given data to a file specified by “filename” in HyDE’s format. The “data” should be in the form produced by saveData. Use Matlab’s “load” function to load a .mat file produced by saveData to bring its data into the workspace, and pass it to the function.

- **writeDataKATE(filename,data)**

  This function works the same as writeDataHyDE, only it writes in KATE’s format.

- **writeAllData**

  This script looks for all saved .mat files in the “Experiments” directory, and calls writeDataHyDE and writeDataKATE for each. HyDE files are places in “Experiments/HyDE” and KATE files are placed in “Experiments/KATE”. These directories must exist for the script to succeed.

- **faultList = getFaultList(componentName)**

  This function returns a cell array of strings containing all the valid fault profiles for the component with the given name. This function is used by LN2GUI and can be used to find out which fault profiles are valid without checking the simulation model.

- **runExperiment(‘Nominal’)**
  runExperiment(component,faultProfile,injectTime)
  runExperiment(component,faultProfile,injectTime,magnitude)
  runExperiment({component faultProfile injectTime magnitude}, …)

  These functions simulate LN2Sim for a particular single or multiple fault scenario, saves the data to a .mat file (by calling saveData), and writes the data as HyDE and KATE files (by calling their respective writeData functions). The output files are named automatically based on the provided arguments. The simulation is then reset back to nominal settings.

  The function can be called in multiple ways. To run the nominal scenario, give the single string argument “’Nominal’”. For single fault experiments, provide “component” which is the component name (see LN2System.xls), “faultProfile” which is the fault profile (see descriptions above, the LN2Library.mdl, LN2GUI, or use the getFaultList function), “injectTime” which is the time of fault injection, and “magnitude” which is the fault parameter (see descriptions above). The magnitude argument does not need to be specified for certain fault profiles, e.g., for the ‘Freeze’ profile. For one or more faults, the function also accepts cell arrays as arguments (specified using “{“ and “}”), where each cell array contains the information to inject a new fault, namely, the component, profile, injection time, and magnitude parameters.

Data in .mat format is saved in the “Experiments” folder. Data in the HyDE format is written in the “Experiments/HyDE” folder, and data in the KATE format is written in the “Experiments/KATE” folder.

The useHyDE configuration variable should be set to 0 before this function is called. The totalTime variable must also be set.

- **runExperiments**

  This is a script that demonstrates how runExperiment and setNoise can be used.

- **setFault(componentName,faultProfile,injectionTime.magnitude)**

  This is the function used internally by runExperiment, and its arguments are similar. This function does not set anything else to nominal first, so multiple calls to setFault can be used to set up a multiple fault scenario.

- **resetToNominal**

  This script sets all components to have the ‘Nominal’ profile.

- **setNoise(sensorNames,mean,variance)**

  This function sets the mean and variance of the sensors with the given names. The sensorNames argument can be a string specifying a single sensor name, a cell array of strings containing multiple sensor names, or the special string ‘all’ which applies the given mean and variance arguments to all sensors. To set only mean, give the empty array, [ ], as the variance argument, and to set only variance, give [ ] as the mean argument.

- **importSystemFile(‘LN2System.xls’)**

  This function creates the LN2System.mat file from LN2System.xls. If LN2System.mat exists and is correct, this function does not need to be called. Make corrections to LN2System.mat (e.g., names of system components in HyDE and KATE) in LN2System.xls, and then regenerate the .mat file by calling importSystemFile on LN2System.xls.

object = findInSystem(list, objectName)

This function returns a struct with the information for the object of the given name in the given list. The list argument can be one of ‘components’, ‘inputs’, and ‘outputs’. An error will be returned if the list argument is not valid or the object with the given name cannot be found in the specified list. This is a helper function used by setFault and setNoise and typically does not need to be called directly. It used LN2System.mat as generated by importSystemFile on LN2System.xls.

A11. DX CONFERENCE / CRASTE CONFERENCE PAPER

The results of the software effort on Rapid Propellant Loading were presented at the 20th International Workshop on Principles of Diagnosis (DX-09) in Stockholm, Sweden, June 14-17 2009. The paper was submitted and accepted as a poster presentation at the conference. This paper has also been accepted for presentation at the 2009 Commercial and Government Responsive Access to Space Technology Exchange (CRASTE) October 26-29, in Dayton, Ohio.

The poster presented at DX-09 (MiniModelDiagnosis_DX09.ppt) and the paper for CRASTE (MiniModelDiagnosis-Final-Submitted.pdf) are included in the publications directory that accompanies this report.

A12. ATTACHMENTS (DIRECTORY LISTING OF SOFTWARE ATTACHMENTS)

```
/Rapid Propellant Loading Software:
  drwxrwxrwx    Hyde
  drwxrwxrwx   KATE
  drwxrwxrwx    Matlab Simulink files
  drwxrwxrwx   Movies
  drwxrwxrwx   Publications
```

APPENDIX B

Applying Model-based Diagnosis to a Rapid Propellant Loading System

Charlie Goodrich*. Sriram Narasimhan, Matthew Daigle**, Walter Hatfield*, Robert Johnson*, Barbara Brown*****

*NASA Kennedy Space Center USA (e-mail: charles.h.goodrich@nasa.gov,walter.h.hatfield@nasa.gov,robert.g.johnson@nasa.gov),
**University of California Santa Cruz USA (e-mail:sriram.narasimhan-1@nasa.gov, matthew.j.daigle@nasa.gov),
***NASA Ames Research Center USA (email:barbara.l.brown@nasa.gov)

Abstract: The overall objective of the US Air Force Research Laboratory (AFRL) Rapid Propellant Loading (RPL) Program is to develop a launch vehicle, payload and ground support equipment that can support a rapid propellant load and launch within one hour. NASA Kennedy Space Center (KSC) has been funded by AFRL to develop hardware and software to demonstrate this capability. The key features of the software would be the ability to recognize and adapt to failures in the physical hardware components, advise operators of equipment faults and workarounds, and put the system in a safe configuration if unable to fly. In December 2008 NASA KSC and NASA Ames Research Center (ARC) demonstrated model-based simulation and diagnosis capabilities for a scaled-down configuration of the RPL hardware. In this paper we present a description of the model-based technologies that were included as part of this demonstration and the results that were achieved. In continuation of this work we are currently testing the technologies on a simulation of the complete RPL system. Later in the year, when the RPL hardware is ready, we will be integrating these technologies with the real-time operation of the system to provide live state estimates. In future years we will be developing the capability to recover from faulty conditions via redundancy and reconfiguration.

1. INTRODUCTION

Rapid propellant loading deals with transferring large amounts (~50,000 gallons) of cryogenic propellant from a storage tank to a vehicle tank. The vehicle tank, though large, is relatively fragile. The vehicle designers have prescribed strict pressure limits for the tank and associated valves. When the cold cryogen liquid first contacts the warm tank and connecting piping, it boils and vaporizes, generating large volumes of gas. This translates to high flow rates and pressures. For this reason, cryogen loading is generally done very slowly using a very structured sequence of events: chilling the cryogen pumps and associated transfer pipes, chilling the vehicle with cold gas, followed by small quantities of cryogen liquid, slow filling of the vehicle with liquid to a predefined level, fast filling the tank to near operating level, slowly topping off the tank to the maximum quantity and incremental replenishment of cryogen liquid that constantly boils off from the vehicle tank prior to launch.

Normally, the process takes about 2-3 hours and a crew of 4 to 8 engineers watches all the equipment on the ground and in the vehicle during this sequence. The engineers are familiar with the physics of cryogenic liquid flow, the pressures and temperatures involved, typical faults that occur with temperature and flow measurements, stuck valves, faulty position indicators, high or low pressure measurements and occasional blown fuses and open circuit breakers. They stand ready to diagnose problems and devise workarounds for faults to complete the loading process and launch the vehicle.

The US Air Force would like to build some of the engineering expertise of the cryogen loading team into the computer system that controls the loading operation. They would like to launch the rocket with a crew of three sergeant-level personnel who may have no engineering experience and limited training in cryogen operations. To this end, the US Air Force Research Laboratory (AFRL) has funded a prototype project at NASA Kennedy Space Center (KSC) to explore rapid cryogenic propellant loading (RPL). The task has two primary goals:

1. Reduce the entire loading operation time to 45 minutes (as opposed to the normal 2-3 hours).
2. Develop software that automates the loading of the vehicle tank with cryogens with minimum operator supervision.

The project is expected to proceed in three phases. In Phase I a small-scale hardware test bed will be built. Supporting software that performs passive diagnosis and some implicit fault accommodation and recovery will also be developed. Phase II will deal with building a larger scale hardware which would include fuel and oxidizer and perform automated umbilical mating and leak detection. Phase III would be a flight demonstration of the developed hardware and software.

The software task of the Phase I project had the following objectives:

1. Development of a preliminary cryogenic test configuration
2. Development of use cases for the preliminary configuration
3. Development of a simulation model of the preliminary cryogenic test configuration (a "mini-model") to conduct tests using the preliminary configuration and use cases
4. Evaluation of the fault detection isolation and recovery (FDIR) application development tools using the mini-model and use cases
5. Development of the RPL Control Architecture
6. Update of the mini-model to the demonstration test bed simulation
7. Testing of the selected Control Architecture with the demonstration test bed simulation.
8. Integration of the Control Architecture with the actual cryogenic test bed hardware
9. Testing of the Control Architecture with the cryogenic hardware

Requirements 1-4 were completed and demonstrated to AFRL in December 2008. In this paper we describe the work that was done in completing the requirements, the setup for the demonstration and some representative scenarios and results.

Our approach to solving this problem was to use model-based technologies where the models themselves would be component-based. The primary reason for this approach was that we would be testing diagnosis algorithms on different systems from the same domain (preliminary configuration, complete RPL configuration, actual hardware). Moreover the models were built while the system configuration was being finalized. Hence component-based models would allow us to compose different system configurations by instantiating appropriate components and connecting them according to the system structure. This approach also allows us to tie anomalous behavior to component faults in a straightforward way.

We selected MATLAB/Simulink® as the language of choice for the simulation model. To support requirement 4, it was decided to evaluate two model-based diagnosis technologies, namely, Hybrid Diagnosis Engine (HyDE) (Narasimhan and Brownston 2007) and Knowledge-based Autonomous Test Engineer (KATE) (Jamieson et al. 1985, Scarl et al. 1987, Goodrich 1995). These two technologies were chosen because of the availability of developers of the two systems. Other technologies may be considered in future years.

The rest of the paper is organized as follows. Section 2 describes the mini-model configuration and faults selected for testing based on use-cases analysis (Requirements 1 and 2). Section 3 describes the MATLAB/Simulink models with fault injection capabilities that were used to generate data for testing (Requirement 3). Section 4 describes HyDE and how models of the mini-model were built in HyDE. Section 5 describes KATE and the efforts to create the mini-model configuration in KATE. Section 6 presents the experimental setup (which was exactly what was demonstrated) that was used to test HyDE and KATE on nominal and faulty data from the simulation and the results from a few representative scenarios.

2. MINI-MODEL CONFIGURATION

2.1 Preliminary Test Configuration

Fig. 1 presents the preliminary cryogenic test configuration (referred to as the mini-model) used for evaluating the model-based diagnosis technologies. The storage tank and vehicle tank both have vent valves (C3 and C4) to regulate the ullage pressure. Three transfer lines connect the tanks to regulate flow between. The main transfer line (top most) and the auxiliary transfer line (second from top) contain pumps that can be controlled to regulate the flows at various rates depending on pump RPM. Both lines contain manually operated valves (G1, G2), remotely operated valves (C1, C2) and flow control valves (A1, A2) to control the flow. The manually and remotely operated valves can be full open or full closed only, whereas the flow control valves can be controlled to be open to any desired percentage. A re-circulation line (bottom most) controlled only by a flow control valve is used to send back excess liquid to the storage tank. The path to the vehicle tank is also controlled by a flow control valve to regulate the ratio of flow going to the vehicle tank and the flow re-circulating to the storage tank.

2.2 Definition of Nominal Operation and Fault Scenarios

Experienced propellant loading engineers stipulated the components and their capacity and likely scenarios for nominal loading operation satisfying the requirements put forth by AFRL. For the sake of the preliminary testing, we decided to ignore the chill down phase, assuming instead that all components were already ready for transferring cryogens. The stages of nominal operations were identified as (i) initialization (ii) slow fill (iii) fast fill (iv) topping (v) replenish and (iv) drain-back in the event of a launch scrub. It was assumed the system would follow this pre-determined sequence except when recovery actions required changes to the sequence.

Engineers also specified likely sensor and component failures based on general knowledge about the components involved and also on prior experience with liquid oxygen loading for the Space Shuttle. The typical faults were identified as sensor (pressure and flow) failures, pump failures and valve (stuck open or stuck closed failures). Some other common failures involving power and data acquisition modules were not included for this initial configuration.

Since we were using a simulation to generate data we had the flexibility of defining which values of the system could be sensed. Typical quantities like flows, pressures, valve positions and tank levels were included corresponding to typical measurement points in the actual hardware. Selected observation points are marked with white circles in Fig. 1.

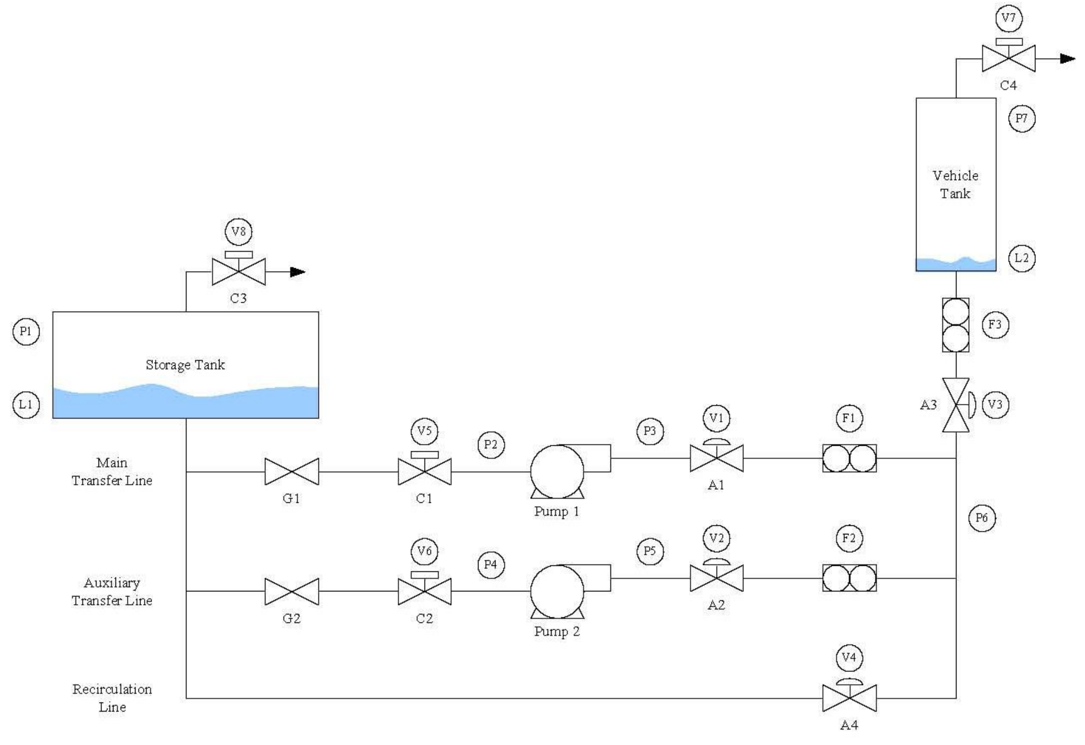


Fig. 1. Mini-model configuration

We were able to take advantage of the flexibility to test the sensitivity of model-based diagnosis tools when faced with different sets of sensor observations. In the future, we hope to use this as a valuable tool to determine sensor placement and diagnosability.

3. MATLAB SIMULATION

In order to evaluate our model-based diagnosis tools, we developed a simulation of the system. The simulation serves as a virtual test bed where we can easily study a large number of fault scenarios to develop our diagnosis models and test our algorithms. Clearly, if an accurate and realistic simulation is developed, then less work is required in migrating diagnosis algorithms to the actual system.

To this end, we developed a physics-based simulation of the system in MATLAB/Simulink. We adopted a component-based modeling paradigm, where parameterized simulation models of generic components including tanks, valves, pipes, pumps, and sensors were developed within a component library. The overall system model is constructed by instantiating the different components from the component library, specifying their parameters, and connecting the components to each other in the appropriate fashion. The top level of the Simulink model which corresponds to the mini-model schematic is illustrated in Fig. 2.

Each component model includes their associated fault modes. For example, a control valve may become stuck at a particular position, or its orifice may become partially blocked. The fault mode, time of fault injection, and fault magnitude (where applicable) can all be specified. In general, each fault mode is mapped to a change in component mode and a fault-dependent magnitude parameter. Because each fault mode is parameterized within the Simulink model, a fault can be injected programmatically (i.e., the fault mode, injection time, and magnitude are specified) either at the beginning of the simulation, or while the simulation is running. The component model of a valve with fault injection capabilities is shown in Fig. 3.

4. HYDE

4.1 Description

Hybrid Diagnosis Engine (HyDE) is a model-based reasoning engine for hybrid (discrete + continuous) diagnosis. HyDE is able to diagnose multiple discrete faults using consistency checking between prediction from hybrid models and sensor observations. HyDE models are component-based and are similar to simulation models. Component models are expressed in terms of behavior in different modes of operation (called locations) including faulty modes with transitions representing the conditions under which the system changes locations. The system model is composed by connecting shared variables between the various components.

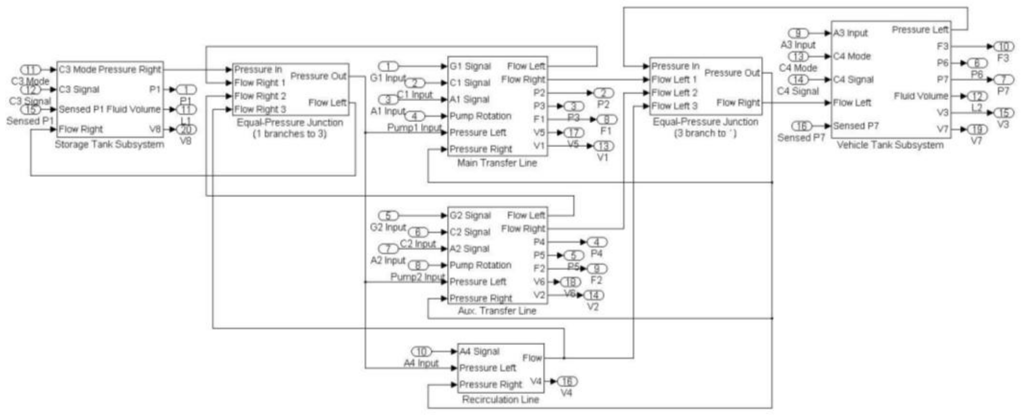


Fig. 2. Simulink model of mini-model configuration

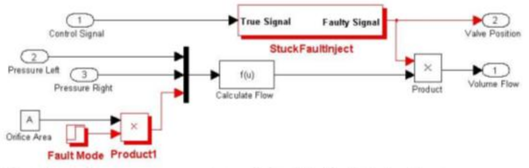


Fig. 3. Valve component model with fault injection capabilities

The HyDE reasoning engine uses the model both for simulation and candidate generation. A set of consistent candidates, each of which may include multiple hypothesized faults, represents the diagnosis. At each time step the candidates are tested for consistency against sensor observations (using simulation) and if found inconsistent new candidates are generated (using candidate generation). For details please refer to (Narasimhan and Brownston 2007).

4.2 HyDE Models of Mini-model Configuration

The HyDE model for the mini-model configuration (Fig. 4) was constructed to look like the Simulink model described earlier. Each component and subsystem in the Simulink model is replicated in the HyDE model. Variables inside each component are also exactly the same as in the Simulink model. This also allowed the connections between components via shared variables to be duplicated in the HyDE model. The behavior within each component had to be modified so as to be represented as hybrid automata (finite state automata with equations in each state). This was done by identifying the modes of operation of the components including the fault modes based on fault injection capabilities. The equations within each mode can be determined by substituting appropriate parameter values corresponding to the different modes as specified in the MATLAB model. The HyDE model of a valve is illustrated in Fig. 5.

The reasoning parameters for HyDE had to be fine-tuned to provide the best performance. The fixed-step Runge-Kutta ODE solver was implemented to simulate behavior across time-steps. There were still some numerical problems since numerical precision used in MATLAB was far superior. As a result the tolerance for fault detection had to be increased (the default was 2%) for some observed variables. For fault isolation the maximum fault size was set to 3 (indicating a maximum of 3 faults in the system). The diagnostic results from HyDE are presented in Section 6.

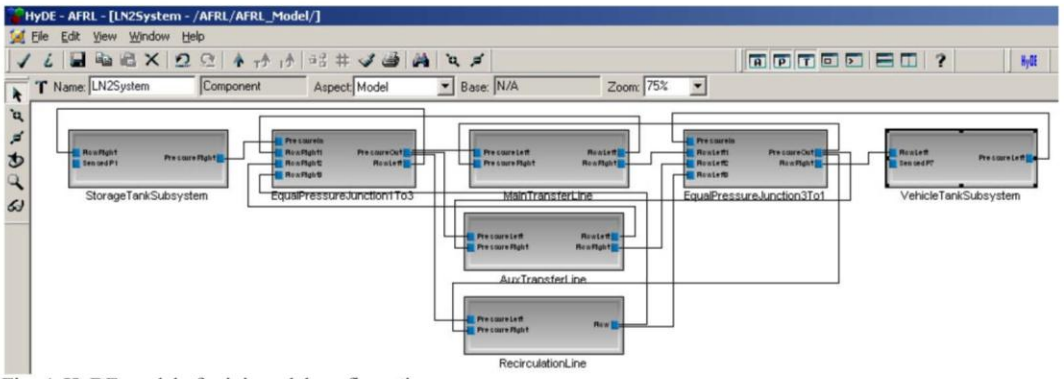


Fig. 4. HyDE model of mini-model configuration

5. KATE

5.1 Description

KATE is a C++ model-based diagnosis system developed by NASA circa 1993 at Kennedy Space Center (Jamieson et al. 1985, Scarl et al. 1987, Goodrich 1995). It was developed to employ Fault Detection, Isolation and Recovery (FDIR) technology to deal with component faults during the Space Shuttle Countdown.

The Shuttle's Launch Processing System (LPS) was programmed to sequence through normal operations and "hold" the countdown if unanticipated sensor readings are encountered. KATE was developed to help Shuttle engineers decide whether such an exception is significant, diagnose the problem, and decide whether or not to continue the countdown.

KATE is a generic software shell for performing model-based FDIR. Each KATE application requires a knowledge-base or model of the system. The same model-based reasoning algorithms are used for each application. The main reasoning systems are:

- Simulation: using a component-based model of a system, the simulation subsystem generates a prediction of behavior of the application system.
- Monitoring: by comparing the predicted sensor values generated by the simulation subsystem with the actual sensor readings of the application system, the monitoring subsystem is able to provide system health status.
- Diagnosis: When a discrepancy between the predicted and actual sensor values is detected, the diagnosis subsystem exploits the structural and functional relationships of the components in the model to determine the failed component(s) that would explain the discrepancy.

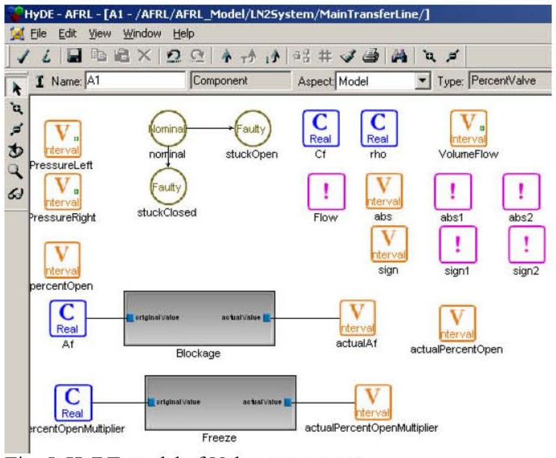
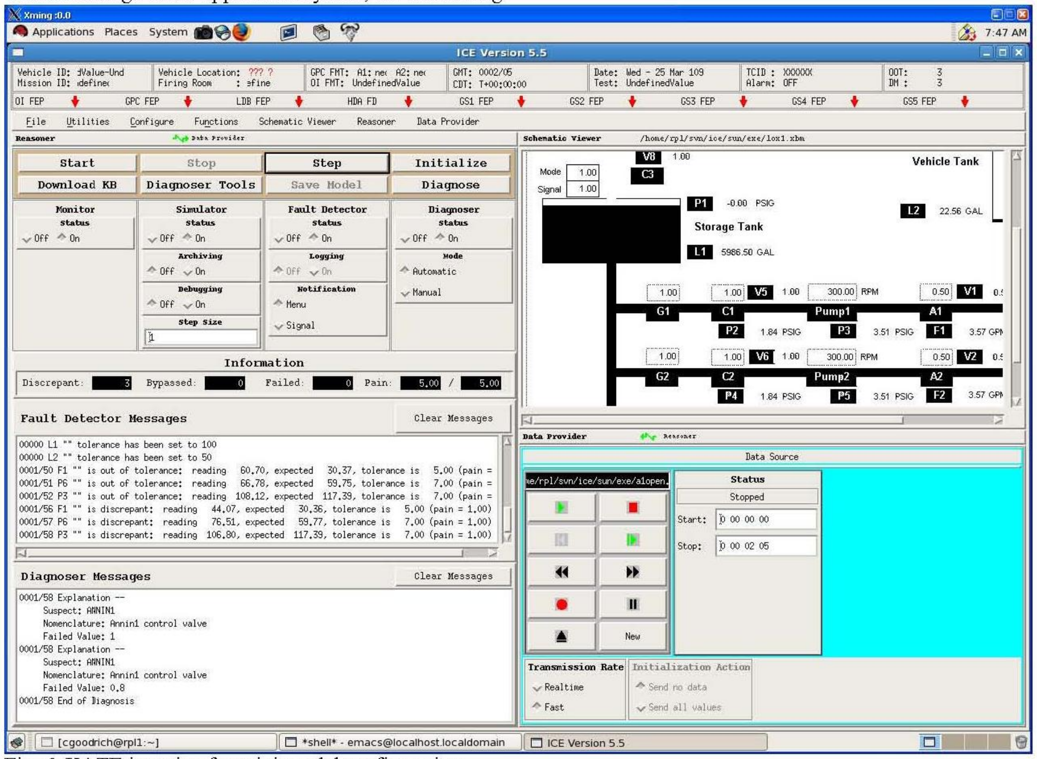


Fig. 5. HyDE model of Valve component



Information

```
Discrepant:   3   Bypassed:   0   Failed:   0   Pain:   5.00  /   5.00
```

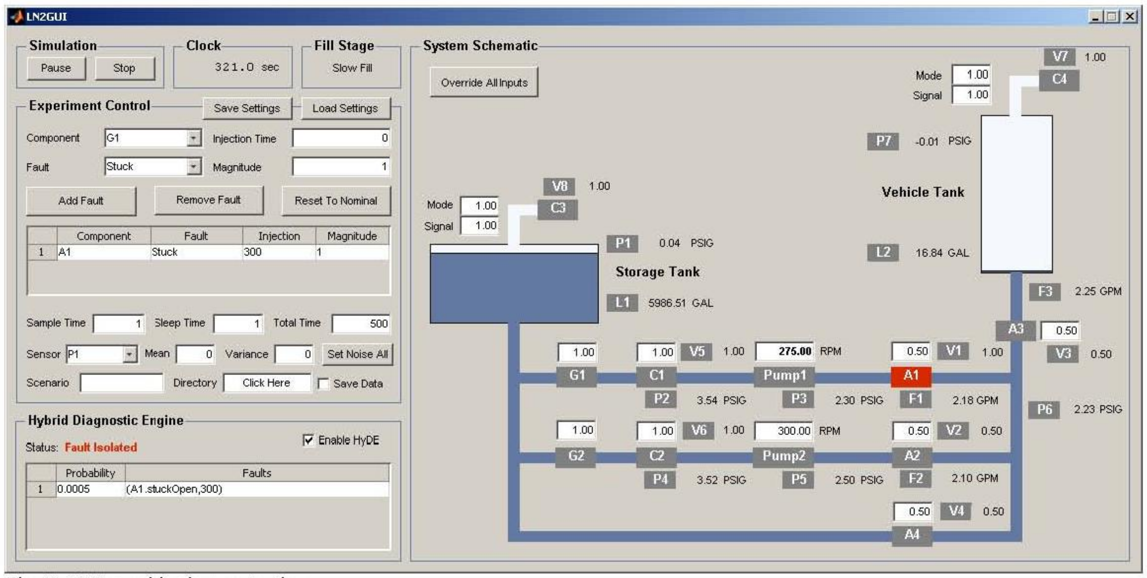
Fault Detector Messages

```
00000 L1 "" tolerance has been set to 100
00000 L2 "" tolerance has been set to 50
0001/50 F1 "" is out of tolerance:  reading   60.70, expected   30.37, tolerance is   5.00 (pain =
0001/51 P6 "" is out of tolerance:  reading   66.78, expected   59.75, tolerance is   7.00 (pain =
0001/52 P3 "" is out of tolerance:  reading  108.12, expected  117.39, tolerance is   7.00 (pain =
0001/56 F1 "" is discrepant:  reading   44.07, expected   30.36, tolerance is   5.00 (pain = 1.00)
0001/57 P6 "" is discrepant:  reading   76.51, expected   59.77, tolerance is   7.00 (pain = 1.00)
0001/58 P3 "" is discrepant:  reading  106.80, expected  117.39, tolerance is   7.00 (pain = 1.00)
```

Diagnoser Messages

```
0001/58 Explanation --
     Suspect: ANNIN1
     Nomenclature: Annin1 control valve
     Failed Value: 1
0001/58 Explanation --
     Suspect: ANNIN1
     Nomenclature: Annin1 control valve
     Failed Value: 0.8
0001/58 End of Diagnosis
```

Fig. 6. KATE in action for mini-model configuration


LN2GUI

Simulation: Pause | Stop

Clock: 321.0 sec

Fill Stage: Slow Fill

Experiment Control

Component: G1 — Injection Time: 0

Fault: Stuck — Magnitude: 1

	Component	Fault	Injection	Magnitude
1	A1	Stuck	300	1

Sample Time: 1 — Sleep Time: 1 — Total Time: 500

Sensor: P1 — Mean: 0 — Variance: 0

Scenario: — Directory: Click Here — ☐ Save Data

Hybrid Diagnostic Engine

Status: Fault Isolated — ☑ Enable HyDE

	Probability	Faults
1	0.0005	(A1.stuckOpen,300)

Fig. 7. GUI used in demonstration

5.2 KATE models of mini-model configuration

The KATE model for the mini-model configuration was built by identifying all the components in the mini-model and considering their likely failure modes. The components include propellant tanks, pumps, valves and sensors. The mini-model uses component behavior defined for the Space Shuttle' KATE LOX application developed in 1986 and refined in 1994. Additional component models were used from a USAF Advanced Launch Operations demonstration developed in 1993. KATE's mini-model was constructed by defining connections between components selected from these pre-existing libraries. Some modifications of the C++ code in the libraries were made to account for the parallel pumping defined in the mini-model.

The resulting "flat file" model simply specified mini-model components from the libraries and connections between the components. KATE was able to diagnose faults by reading data generated by the Matlab simulation. The diagnostic results from KATE are presented in the next section.

6. EXPERIMENTAL SETUP, DEMO AND SCENARIOS

In order to demonstrate the simulation and diagnosis capabilities we created command line and visual interfaces. Using the command line interface, the user can specify an experiment, parameterized by a set of faults to inject, the amount of noise for each sensor, and the sample time of the sensors. The simulation is then automatically executed with these parameters, and the resulting input and output data is written to files for input to the diagnostic tools. Currently we support the creation of data files recognized by HyDE and KATE tools.

The visual interface, a MATLAB GUI linked to the Simulink model, provides an overview schematic of the mini-model configuration and allows the user to setup the experiment. It sends the user specified parameters to Simulink and runs the simulation for the specified time. All input and output values are updated on the GUI as the simulation progresses. The user can also change the inputs online to recover from failures, and click on any variable in the GUI to see a time series plot. Fig. 7 shows the GUI in action.

In addition, we have also implemented an interface from the GUI/Simulink to HyDE. This was achieved by creating a MEX function in C++ that acts as the interface. The MEX function, which is linked to HyDE static library, creates an instance of HyDE, loads the appropriate model and configuration parameters into HyDE and then executes HyDE API functions in step with the simulation. Command and sensor data from the simulation is sent to HyDE at each time step. The diagnosis candidates from HyDE are listed in the GUI (lower left side). When the user clicks on each candidate, faulty components are colored red in the schematic to provide a visual indication.

Due to lack of time, an interface between the GUI and KATE was not created. We chose instead to exercise KATE using the data files generated (either using the command line interface or the GUI).

We present and discuss results from 4 scenarios. These were the same 4 scenarios used in the demonstration. The nominal operational sequence was selected to mimic the stages of a realistic cryogenic propellant loading. The first scenario was the complete nominal scenario with the goal of validating the realism of the simulated data. Three fault scenarios were selected based on use case analysis that determined some common failures (discussed in Section 2). Note that HyDE did not use the valve position sensors in any of the scenarios.

The first fault scenario was a flow control valve (A1) stuck open during the slow fill stage (at 300 s). The flow control valves can be controlled on a continuous scale from 0 to 1 to regulate the flow. There is a flow control valve in the main and auxiliary lines and in the common discharge line leading to the vehicle tank (Fig. 1). When one of these valves is stuck open it is necessary to regulate the flow in that line using other means. HyDE and KATE are able to diagnose this situation right after observations for time 300 s are available. We determined that the recovery action for this fault was to regulate the flow by decreasing the Pump 1 speed to 275 RPM during slow fill to achieve the required net flow to the vehicle tank. Fig. 8 shows the profiles of some key measurements for this scenario.

The second fault scenario was a vent valve failure (stuck closed). The operation of this valve is modeled as an autonomous mode change (is not externally commanded). Its nominal behavior is to regulate the ullage pressure of the vehicle tank between 2 and 7 psig. Because the state of the valve is not directly commanded, this adds a level of difficulty to the reasoning process. Further, this was a critical fault scenario because if nothing was done, then the ullage pressure in the tank would keep increasing, resulting in catastrophic failure. The other interesting feature of this failure is that it can only be detected when the vent valve is predicted to be in open state since the behavior when the valve is in the closed state and the stuck closed state is identical. The scenario chosen failed the valve when it is in the closed state (1100 s). At 1254 s, the ullage pressure (P7) exceeds 7 psig, and the valve should have opened. At this point, HyDE and KATE are able to diagnose that the valve is stuck closed, although they cannot determine the time at which the failure occurred since it may have occurred at any time during which the valve was in the closed state. In this situation, the system is unsafe and loading operations must be aborted. Therefore, the recovery action is to drain the propellant back from the vehicle tank to the storage tank by shutting down the pumps and opening up all the valves. Fig. 9 illustrates the profiles of some key measurements in this scenario.

The third fault scenario was a double fault case where a pump (Pump 1) and a down stream flow sensor (F1) failed. Only HyDE was run on this scenario since KATE currently does not support multiple fault diagnosis. Additionally, we also decided to use only flow sensors to demonstrate the sensor fusion capabilities of HyDE when limited sensing is available. When the pump in the main transfer line fails, it is expected that the flow sensor in the main transfer line would indicate this failure. However, if the flow sensor has failed, it is necessary to use other sensors (namely flow sensors in the auxiliary transfer line and the vehicle tank subsystem) to diagnose the pump failure. HyDE is able to do exactly that albeit taking a couple of time-steps since the effects on the other flow sensors are not immediately seen. The recovery action for this situation was determined to be increasing the pump RPM in the auxiliary transfer line to 375 RPM and opening up the flow control valve in that line completely, in order to regain the nominal flow into the vehicle tank. Fig. 10 shows the profiles of some key measurements for this scenario.

Although we presented only three fault scenarios, we tested HyDE and KATE extensively by varying the number of faults injected (only single faults for KATE), the time at which the faults are injected, the sensor noise, and which sensor observations were available to the diagnosis engines. For KATE we tested only the scenarios described earlier in this section. However for HyDE we performed more systematic tests. There were 25 components that could be faulted with approximately 2 fault modes each. This in combination with 6 operating regions (Section 2.2) gave us 300 interesting single fault cases. We also tested ~100 double fault cases, all of which were a combination of a component and a sensor fault. All the single faults were diagnosed but there were 2 double fault cases that could not be diagnosed by HyDE. In one case a combination of a vent valve failure and corresponding pressure sensor failure resulted in fault effects not being seen until a few time steps later. In another scenario, HyDE was not able to distinguish between manual operated valve failure and remotely operated valve failure (position sensors were not used) and picked the wrong one because it had higher prior probability of failure.

7. CONCLUSIONS AND FUTURE WORK

In this paper we presented an application of model-based simulation and diagnosis work to a scaled-down version of a rapid propellant loading system. We demonstrated the detection and isolation of some common failure scenarios. In the process we also developed a component-based simulation in MATLAB and showed its usefulness in testing and evaluating FDIR applications.

The next steps in the project involve the completion of requirements 5-9 detailed in the introduction. We are in the process of developing a software architecture based on the Internet Communication Engine (ICE) (Henning 2004) to support requirement 5. ICE uses a "publish-subscribe" protocol for communication. In the architecture which was adopted from the Advance Diagnostics and Prognostics Test bed (ADAPT) (Poll et al. 2007), the MATLAB simulation will subscribe to the user inputs (including commands) and publish sensor data. The operation of the simulation is similar to the Virtual ADAPT simulation. HyDE and KATE will subscribe to the user inputs and sensor data and publish diagnosis results. The MATLAB model is being updated to reflect the hardware test bed being constructed (requirement 6). HyDE and KATE models will be built for the new configuration and a demonstration of the results is scheduled for late April 2009 (Requirement 7). When the test bed hardware has been built, the control architecture can be easily modified by replacing the MATLAB simulation with the LABVIEW module performing the data acquisition on the hardware (requirement 8). HyDE and KATE models can then be tested against data generated from the actual hardware (requirement 9).

Phase II of the project would deal with the implementation of automated synthesis of recovery actions based on the current state of the system. This may have to be interleaved with the diagnosis process for 2 reasons: (i) the diagnosis engine frequently reports ambiguities and recovery actions may have

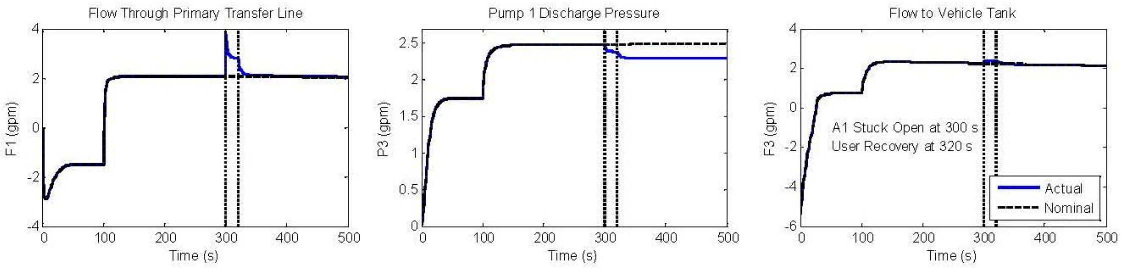


Fig. 8. Fault Scenario 1 with stuck control valve

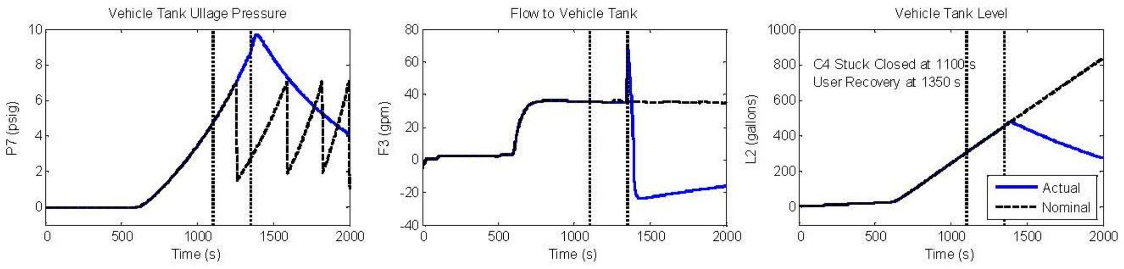


Fig. 9. Fault Scenario 2 with stuck vent valve

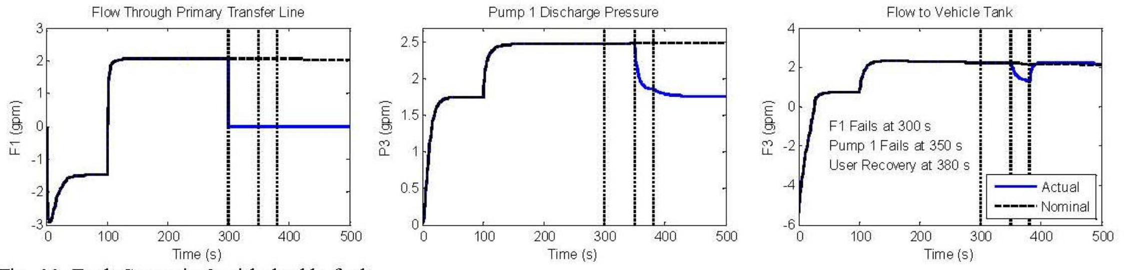


Fig. 10. Fault Scenario 3 with double fault

to be taken to avoid catastrophic effects of any faults reported, if time is taken to wait for further disambiguation (ii) in some cases recovery actions may be used to provide information helpful in disambiguating faults.

ACKNOWLEDGMENTS

The authors are indebted to the RPL development team at NASA Kennedy Space Center, FL and the Discovery and Systems Health Area at NASA Ames Research Center whose work made this publication possible including Bradley Burns, Bert Cummings, Bob Ferrell, Jesse Goerz, Kevin Jumper, Jan Lomness, Ann Patterson-Hine, Jose' Perotti, David Richter and Jared Sass.

REFERENCES

S. Narasimhan and L. Brownston. HyDE – A General Framework for Stochastic and Hybrid Model-based Diagnosis. In *Proc. 18th International Workshop on Principles of Diagnosis (DX '07),* pages 162-169, Nashville, USA, May 2007.

J. Jamieson, E. Scarl, and C.I. Delaune. A Knowledge Based Expert System for Propellant System Monitoring at the Kennedy Space Center. In *Proceedings of the 22nd Space Congress.* Cocoa Beach, FL, Apr 1985.

E.A. Scarl, J.R. Jamieson, and C.I. Delaune. Diagnosis and Sensor Validation through Knowledge of Structure and Function, *IEEE Transactions on Systems, Man, and Cybernetics,* 17(3):360-368, May/June 1987.

C. Goodrich. A Method for Diagnosing Time Dependent Faults using Model-Based Reasoning Systems, *FLAIRS (Florida AI Research Symposium).*

M. Henning. A New Approach to Object-Oriented Middleware. *IEEE Internet Computing*, 8(1): 66-75, 2004.

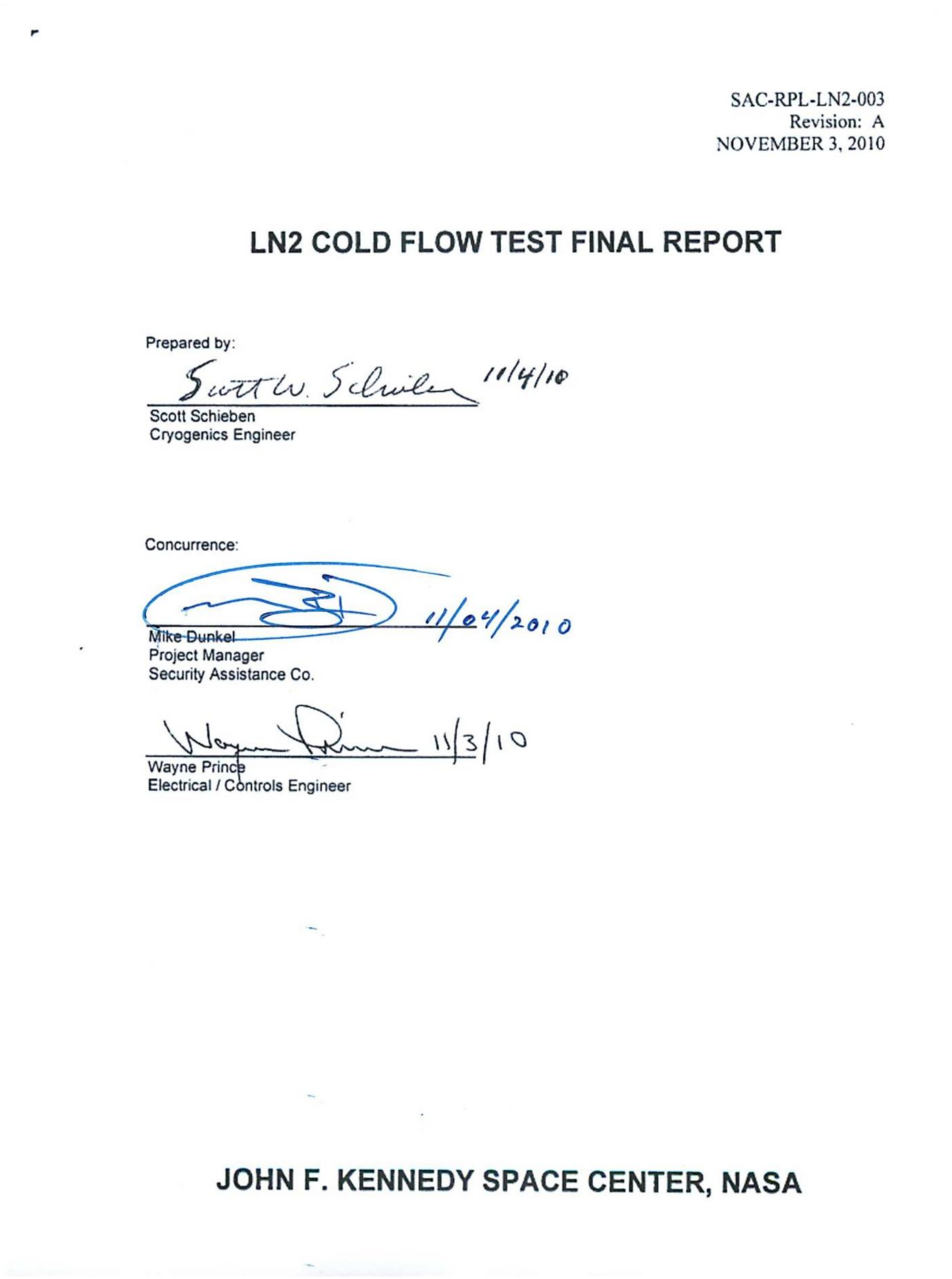
S. Poll S. A. Patterson-Hine, I. Roychoudhury, M. Daigle, G. Biswas, et al. Evaluation, Selection, and Application of Model-based Diagnosis Tools and Approaches. *AIAA-2007-2941. AIAA Infotech@Aerospace*, Rohnert Park, CA, May 2007.

APPENDIX C

RAPID PROPELLANT LOADING (RPL) CRYOGENIC TANKING DEMONSTRATION

LIQUID NITROGEN (LN2) COLD FLOW TEST FINAL REPORT

SAC-RPL-LN2-003



SAC-RPL-LN2-003
Revision: A
NOVEMBER 3, 2010

LN2 COLD FLOW TEST FINAL REPORT

Prepared by:

[illegible] 11/4/10
Scott Schieben
Cryogenics Engineer

Concurrence:

[illegible] 11/04/2010
Mike Dunkel
Project Manager
Security Assistance Co.

[illegible] 11/3/10
Wayne Prince
Electrical / Controls Engineer

JOHN F. KENNEDY SPACE CENTER, NASA

RECORD OF REVISIONS/CHANGES

REV LTR	CHANGE NO.	DESCRIPTION	DATE
Basic		Initial Release	November 1, 2010

TABLE OF CONTENTS

LIQUID NITROGEN COLD FLOW TEST

C1. Reference Information

C1.1 Referenced Documents

Drawing No.	Description	Revision
SAC-LN2-RPL-001	Test Setup Leak Check	LI
SAC-LN2-RPL-002	Rapid Propellant Loading (RPL) Cryogenic Tanking Demonstration – Liquid Nitrogen (LN2) Cold Flow Test	LI

C2. Scope

On August 24th 2010 a cryogenic loading demonstration was performed on the Rapid Propellant Loading (RPL) system. The test was accomplished by transferring Liquid Nitrogen (LN2) from a portable tanker through a cold flow control system and into an instrumented composite test tank, using varied cryogenic transfer techniques. This report documents the results from that cryogenic loading demonstration.

C3. Test description

The demonstration consisted of 3 different cryogenic loading techniques:

1. Tank loading using a bottom fill technique (using only a 2” transfer line into the bottom of the test tank).

2. Tank loading using a top fill technique (through a spray nozzle located in the top of the test tank).

3. Tank loading using a combined top fill and bottom fill technique (using both the bottom fill line and the top spray nozzle).

All testing was performed in accordance with procedure# SAC-LN2-RPL-002 “Rapid Propellant Loading (RPL) Cryogenic Tanking Demonstration – Liquid Nitrogen (LN2) Cold Flow Test.”

C4. Test setup

The LN2 cold flow test setup mechanical schematic is shown in Figure C-1 below. The mechanical setup consisted of the following major sub-assemblies:

- Pneumatically-actuated 2-position (open/closed) control valves for the top fill and vent systems
- A variable-position flow control valve for the bottom fill system
- Regulated gaseous nitrogen pneumatic supplies for these valves
- An interface to connect the system to a portable LN2 tanker
- Fill, vent & drain line piping for the test tank
- A delta pressure / orifice apparatus to measure flow-rate in the bottom fill system
- A circular spray "nozzle" assembly located in the top of the test tank.

The LN2 cold flow test setup electro-mechanical schematic is shown in Figure C-2 below. The electrical / control system consisted of the following major sub-assemblies:

- Solenoid valves to provide gaseous nitrogen actuation pressure for the top fill and vent pneumatic valves
- A current-to-air pneumatic controller for the bottom fill flow control valve
- Valve position indicators
- Pressure measurements for the flow-rate orifice
- An instrumentation mast inside the test tank containing numerous temperature sensors
- External temperature sensors on the exterior of the tank.

All of the instrumentation and controls were operated via a control coupler apparatus (note: this apparatus is incorrectly identified as a PLC on the electro-mechanical schematic).

Appendix C-A contains photographs of the LN2 cold flow test setup; these photos were taken during assembly and pre-test checkout of the apparatus. After completion of assembly and checkout, the apparatus was shipped to the NASA Fire Protection Training Area on Static Test Road, where all cryogenic testing was performed.

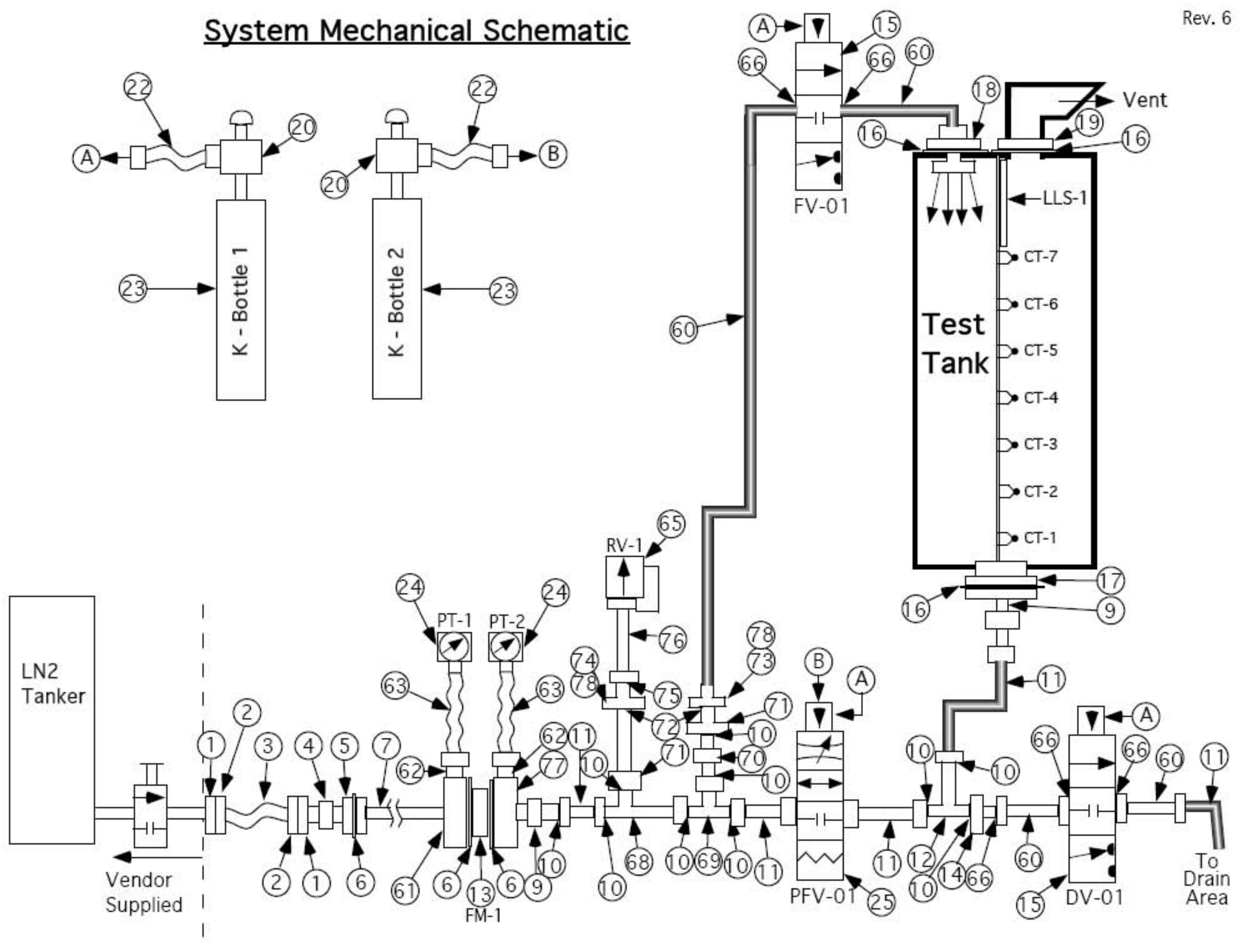


Figure C-1
System Mechanical Schematic

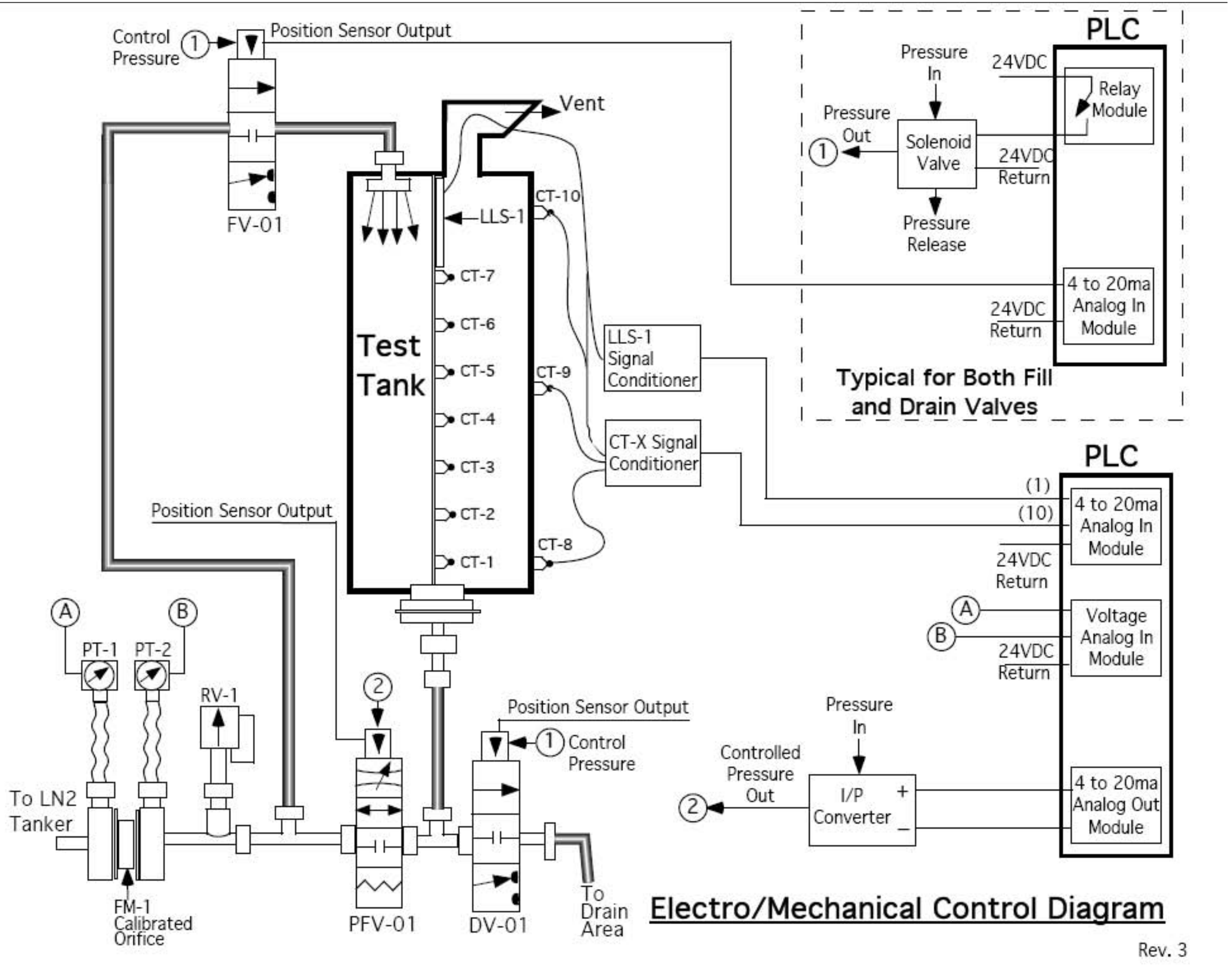


Figure C-2
Electro-Mechanical Control Diagram

C5. Test Results and observations

C5.1 Run 1: Tank Loading Using Bottom Fill Technique

The Liquid Nitrogen (LN2) supply trailer was pressurized to approximately 35 psig and LN2 flow was initiated into the cold flow apparatus. Initial chill-down of the control skid, as evidenced by steady LN2 flow out of drain valve DV-01 took approximately 3 minutes. At this time, DV-01 and top fill control valve FV-01 were closed and liquid was routed into the test tank via bottom fill flow control valve PFV-01.

DV-01 was initially set to a mid-range position (approximately 50% open) and subsequently opened more fully as tank loading progressed. The liquid level in the test tank was initially monitored by visually watching the ice/frost level on the exterior of the tank. The exterior tank temperature sensors, located at the bottom, middle and top of the tank, were also monitored for indications of cryogenic temperatures. When liquid nitrogen reached the upper levels of the tank, it was monitored on the temperature sensors located on the instrumentation mast. A liquid level capacitance probe, also located on the mast, was also monitored during tanking operations, but this probe never indicated liquid during tanking.

It became evident very early in the tanking operation that the tanking system was operating near its capacity, and that chill-down of the test tank was preceding slower than anticipated. The following specific conditions were observed:

- It was apparent that the test tank was experiencing significant liquid boil-off conditions, probably due to a combination of warm ambient temperatures (approaching 90 deg. f) and the lack of any external insulation on the tank.

- LN2 flow into the tank was being limited by the orifice being used to measure LN2 flow-rate in the bottom fill line. The 1-inch diameter orifice in the 2-inch diameter bottom fill line was evidently limiting the LN2 flow-rate.

As a result of the above conditions, it was apparent that successfully filling the tank to the 100% level would require some adjustments to the loading procedure. The following steps were taken to maximize LN2 flow-rate into the tank:

- LN2 tanker pressure was increased to the maximum tanker operating pressure of 40 psig

- Bottom fill flow control valve PFV-01 was opened fully (100%)

- Top fill control valve FV-01 was opened and LN2 was introduced into the top of the tank, in order to facilitate tank chill-down.

The above steps proved to be successful in facilitating additional LN2 flow into the test tank, and liquid levels eventually reached the top of the tank, as monitored on the tank temperature sensors. However, as stated previously, no liquid was ever observed on the capacitance liquid level sensor. Additionally, no liquid was ever observed at the outlet of the tank vent duct at the top of the vessel.

The elapsed time from the end of chill-down to the determination that the test tank was at the 100% full level was approximately 33 minutes. It is clear that if the loading procedure adjustments discussed previously were employed earlier, the tanking timeline could be significantly reduced.

Appendix C-B contains photographs of the RPL LN2 cold flow test setup; these photos were taken during Run 1 (Bottom Fill – described above) and Run 2 (Combined Top and Bottom Fill – described below).

Appendix C-C contains graphical representations of LN2 tanking data for both Run 1 (Bottom Fill – described above) and Run 2 (Combined Top and Bottom Fill – described below).

C5.2 Run 2: Tank Loading Using Combined Top and Bottom Fill Techniques

Following Run 1 (Tank Bottom Fill), the test tank was drained completely and allowed to warm to ambient temperature. Prior to the next tanking operation, the NASA / Contractor test team conducted an informal technical review meeting and determined that the next planned tanking operation "Tank Loading Using Top Fill Technique" would not be required. This determination was made due to the fact that the top fill line had been utilized during the latter part of Run 1, and adequate data had been obtained during that operation. A decision was made to proceed directly to the "Tank Loading Using Combined Top Fill / Bottom Fill Technique" sequence of the test procedure.

The test team also determined that it would be advantageous to remove the 1-inch flow measurement orifice from the system prior to Run 2, in order to maximize the LN2 flow-rate into the test tank. This work was accomplished and the flow-rate measurement was inoperative during Run 2.

Initial chill-down of the control skid was similar to Run 1, with a steady LN2 flow out of drain valve DV-01 occurring approximately 3 minutes after the start of LN2 flow. At this time, drain valve DV-01 was closed but top fill control valve FV-01 remained open to facilitate tank chill-down. Liquid continued to be routed into the test tank via bottom fill flow control valve PFV-01.

As a result of the lessons learned and procedural adjustments implemented during Run 1, the Run 2 tanking timelines were significantly shorter. The elapsed time from the end of chill-down to the determination that the test tank was at the 100% full level was approximately 20 minutes.

The same liquid level monitoring procedure employed during Run 1 was once again utilized during Run 2. The liquid level in the test tank was monitored by visually watching the ice/frost level on the exterior of the tank, the exterior tank temperature sensors, located at the bottom, middle and top of the tank, were monitored for indications of cryogenic temperatures, and the temperature sensors located on the instrumentation mast were monitored. Unlike Run 1, the liquid level capacitance probe located on the mast momentarily indicated liquid when the test tank was completely full, although at lower quantities than anticipated.

A final significant difference from Run 1 was that the increased LN2 flow-rates allowed the control system to completely overcome the significant LN2 boil-off conditions. Therefore, when the tank reached the 100% full level, traces of LN2 were evident at the tank vent duct outlet, which provided a further verification that the tank was indeed full.

Appendix C-B contains photographs of the RPL LN2 cold flow test setup; these photos were taken during Run 1 (Bottom Fill – described above) and Run 2 (Combined Top and Bottom Fill – described below).

Appendix C-C contains graphical representations of LN2 tanking data for both Run 1 (Bottom Fill – described above) and Run 2 (Combined Top and Bottom Fill – described below).

Appendix C-A
LN2 Cold flow apparatus assembly photographs

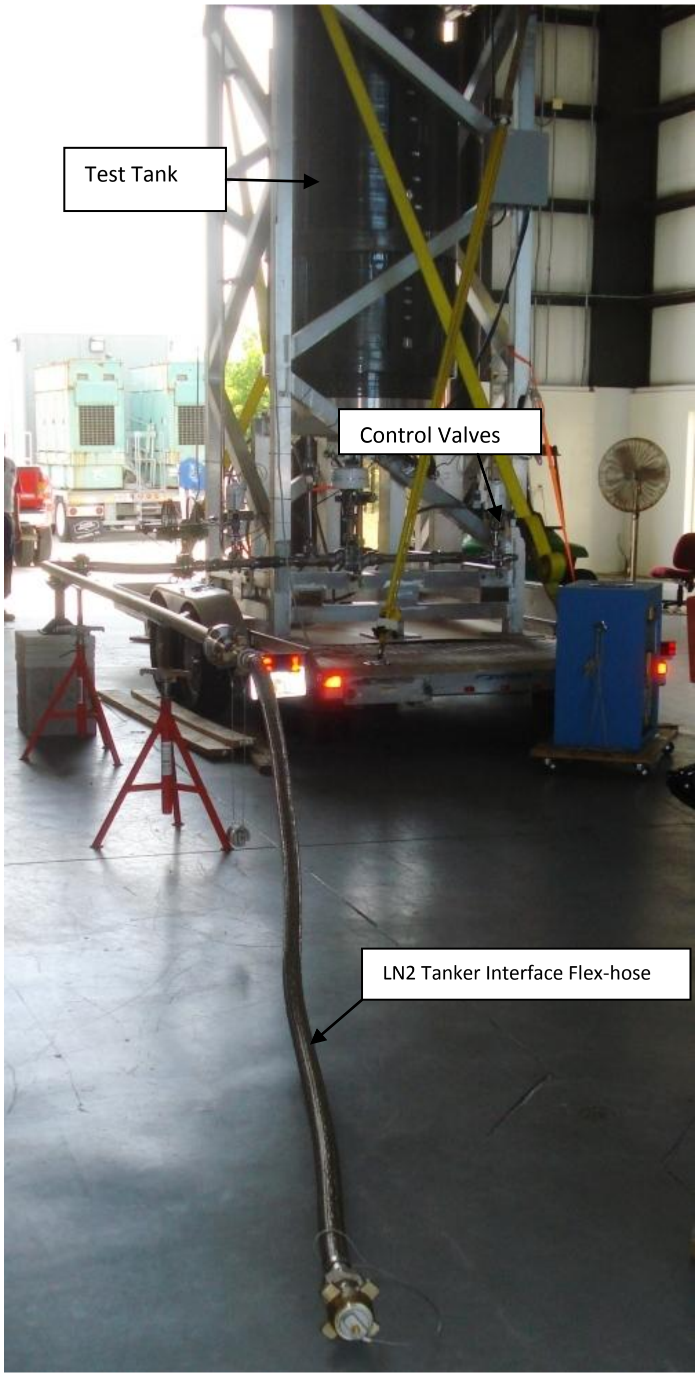


Figure C-3
Test Apparatus – Front View

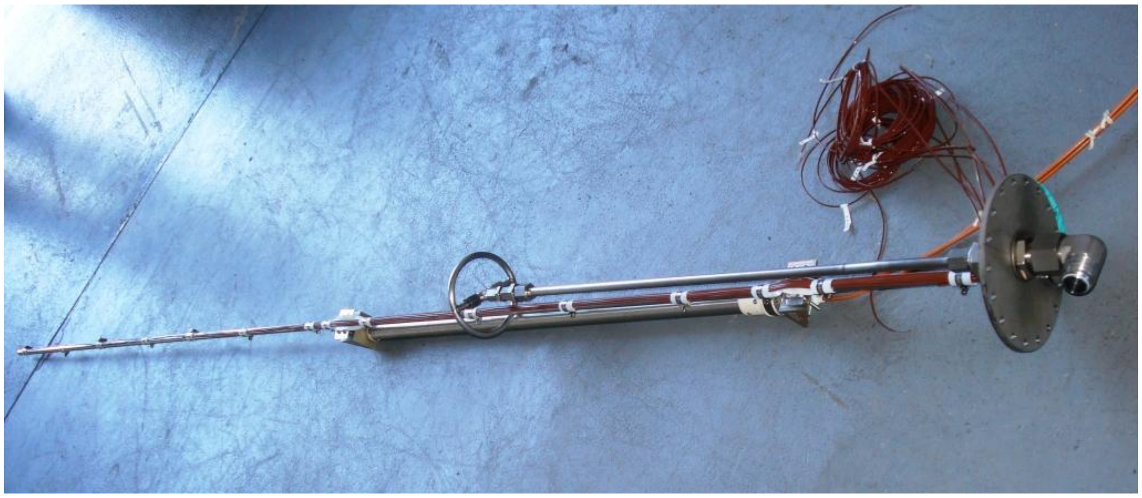


Figure C-4
Instrumentation Mast

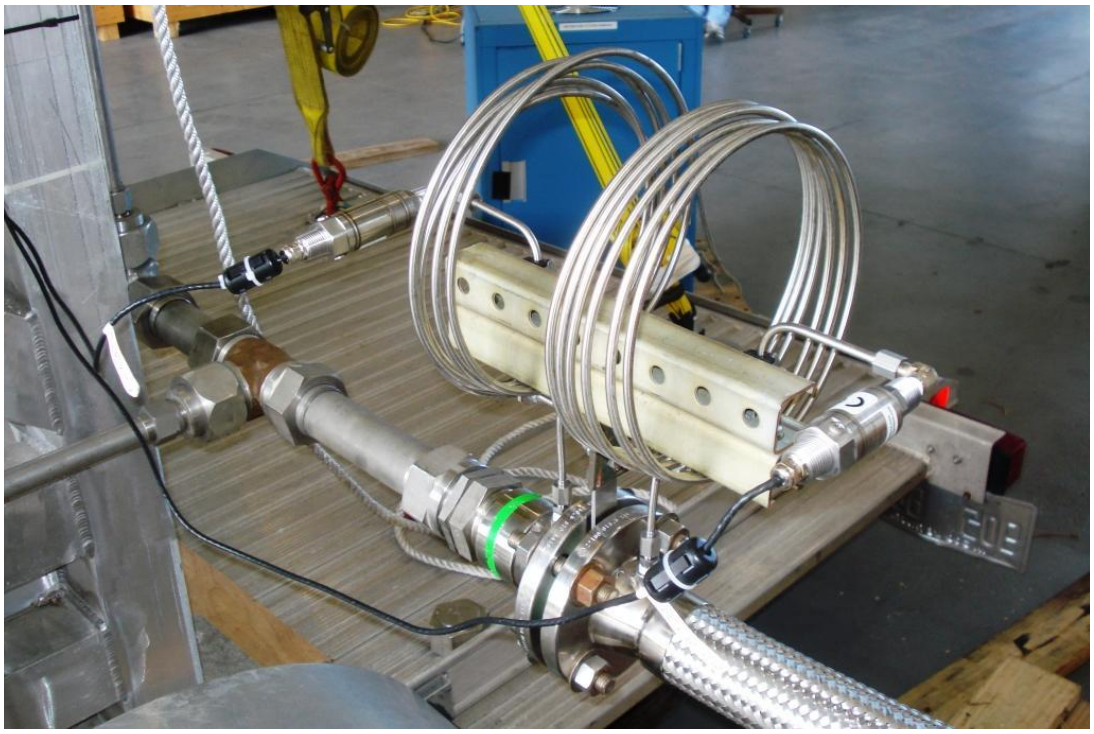


Figure C-5
Flow-rate Orifice

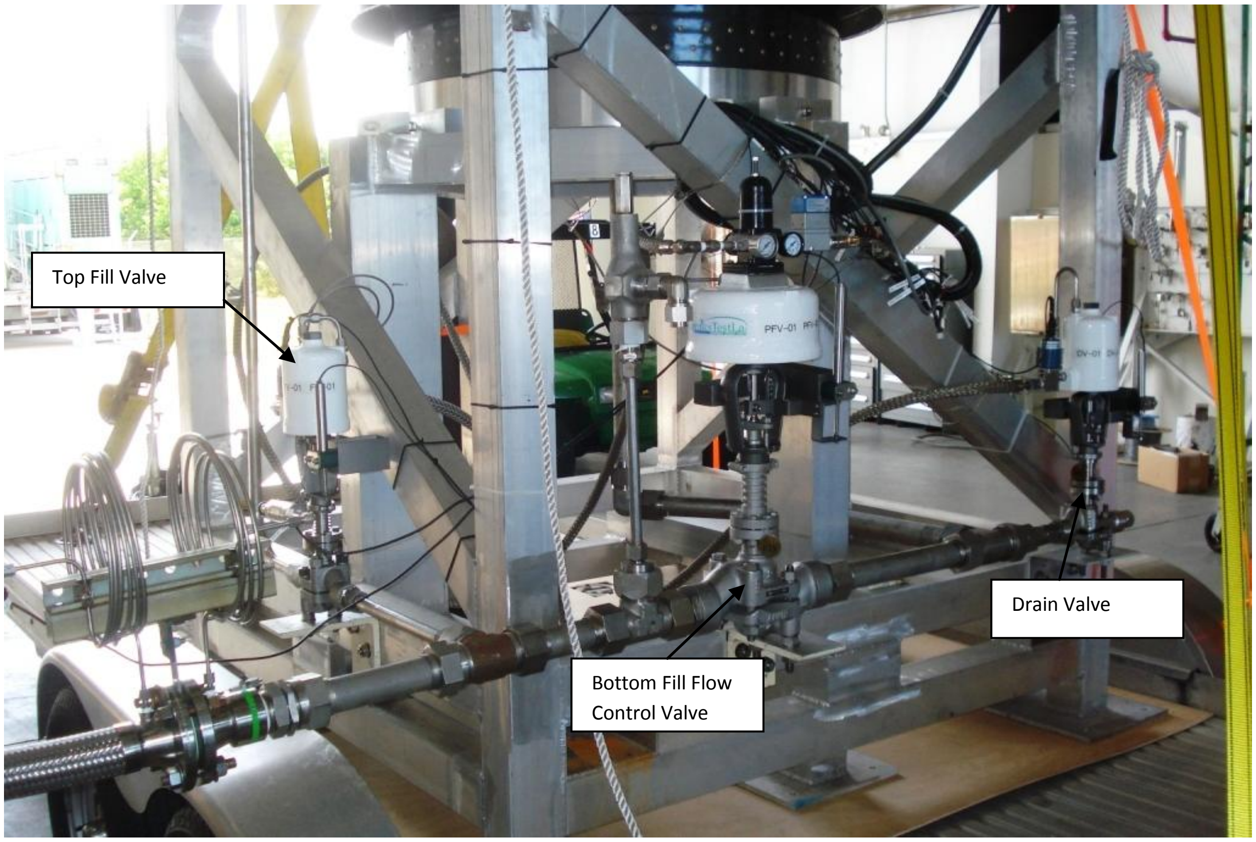


Figure C-6
Test Apparatus Close-up

Appendix C-B
LN2 tanking photographs

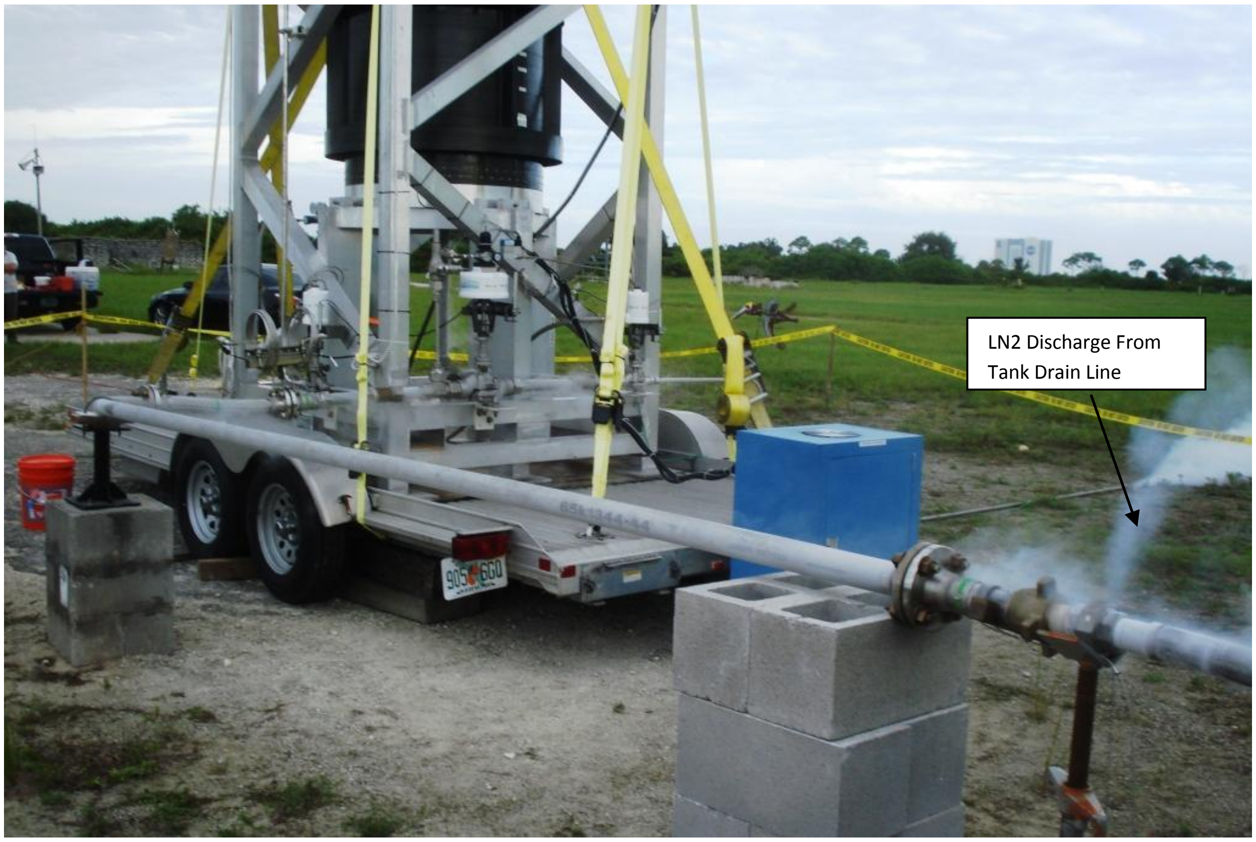


Figure C-7
Start Of Chill-down, Run 1 (Bottom Fill)

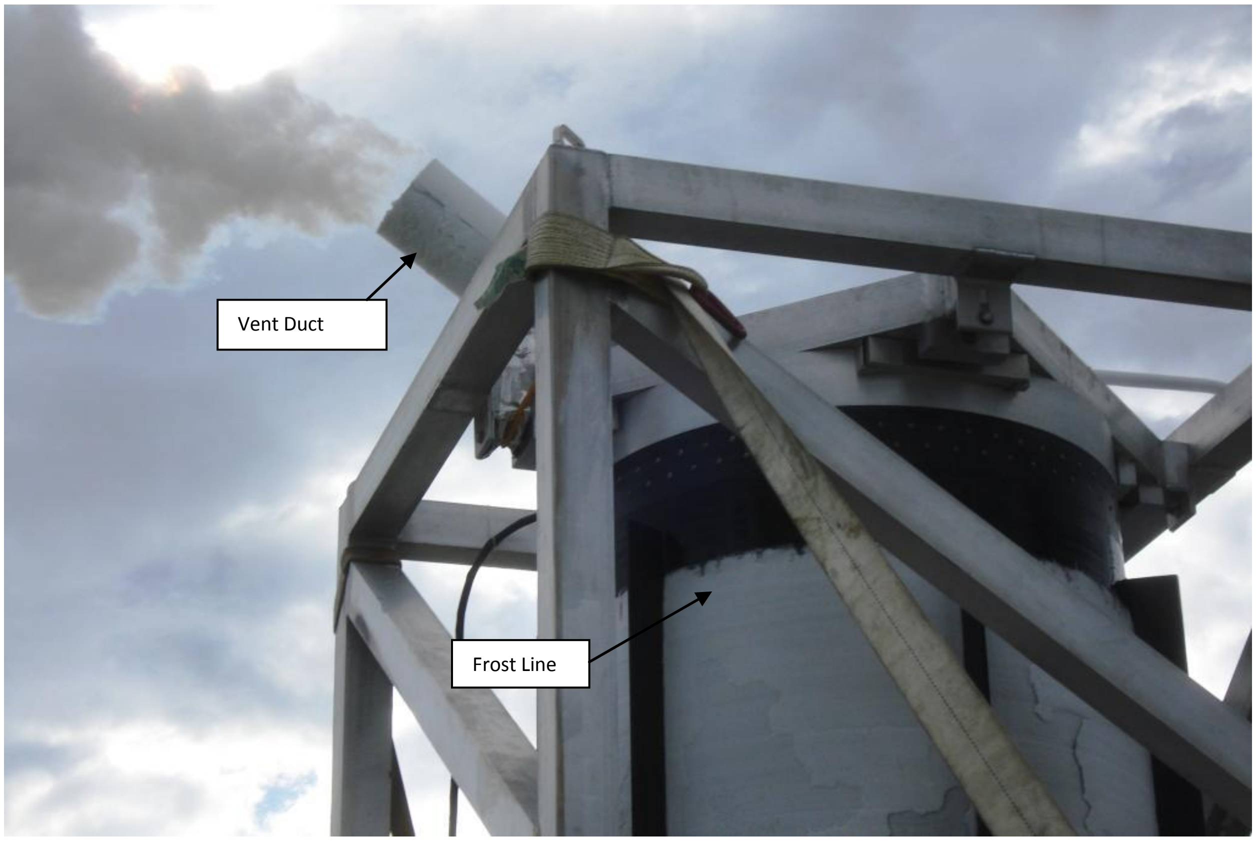


Figure C-8
Tank Venting, Run 1

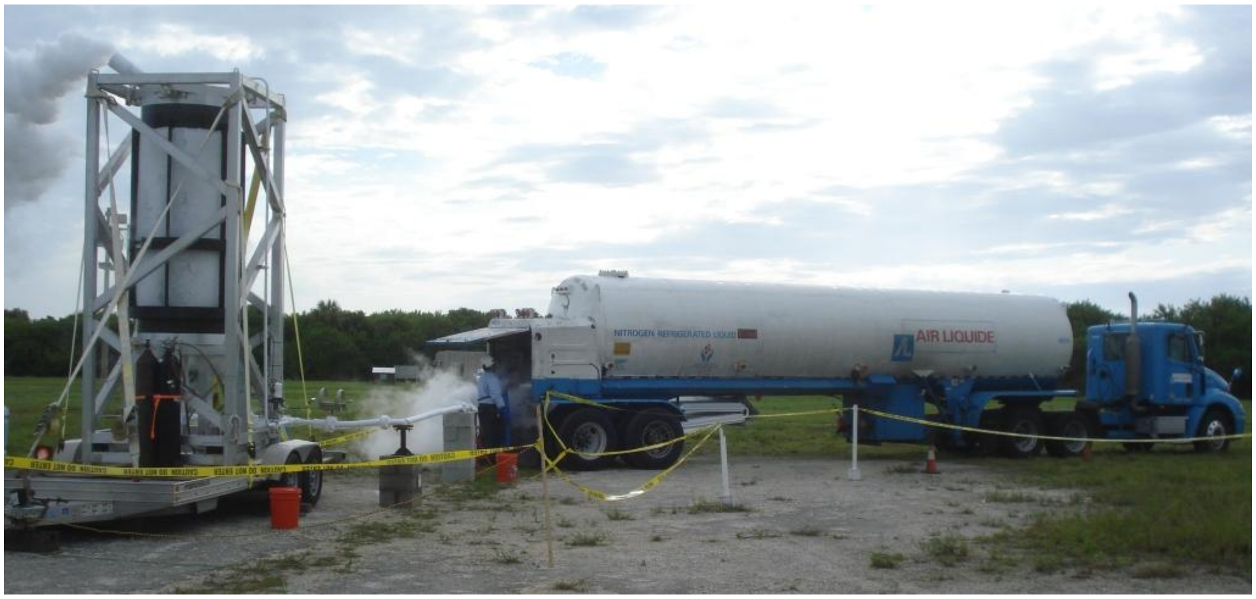


Figure C-9
Test Apparatus Overview

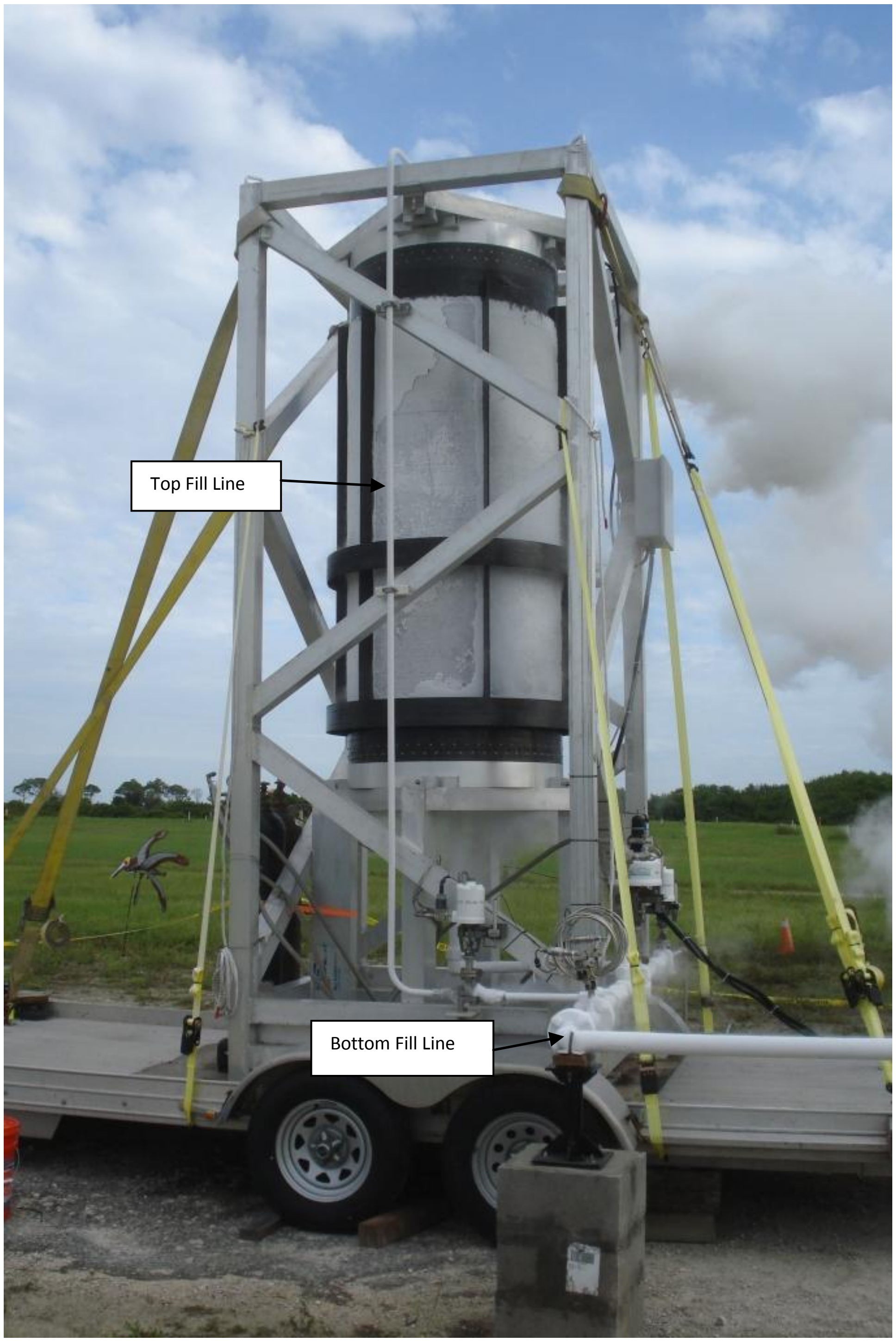


Figure C-10
Combined Top and Bottom Fill

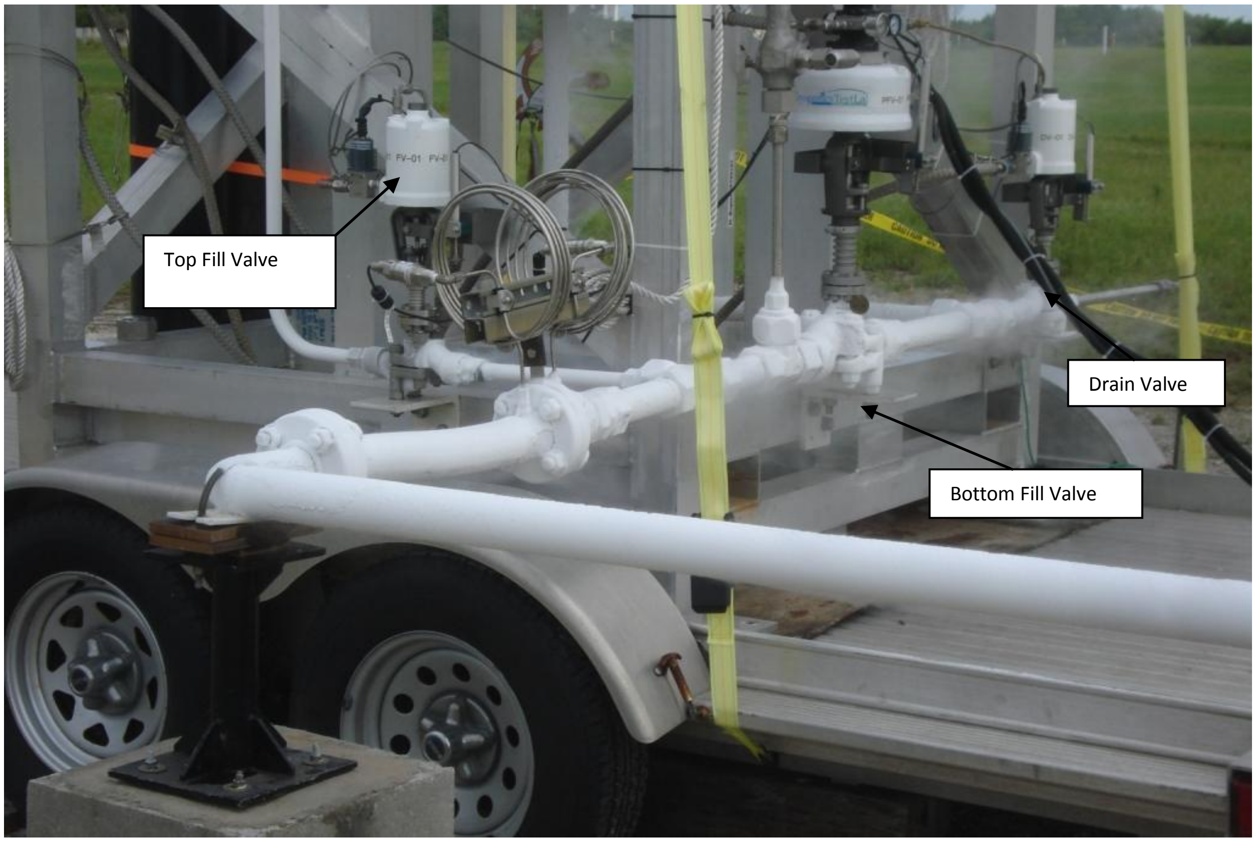


Figure C-11
Control Valves, Top and Bottom Fill

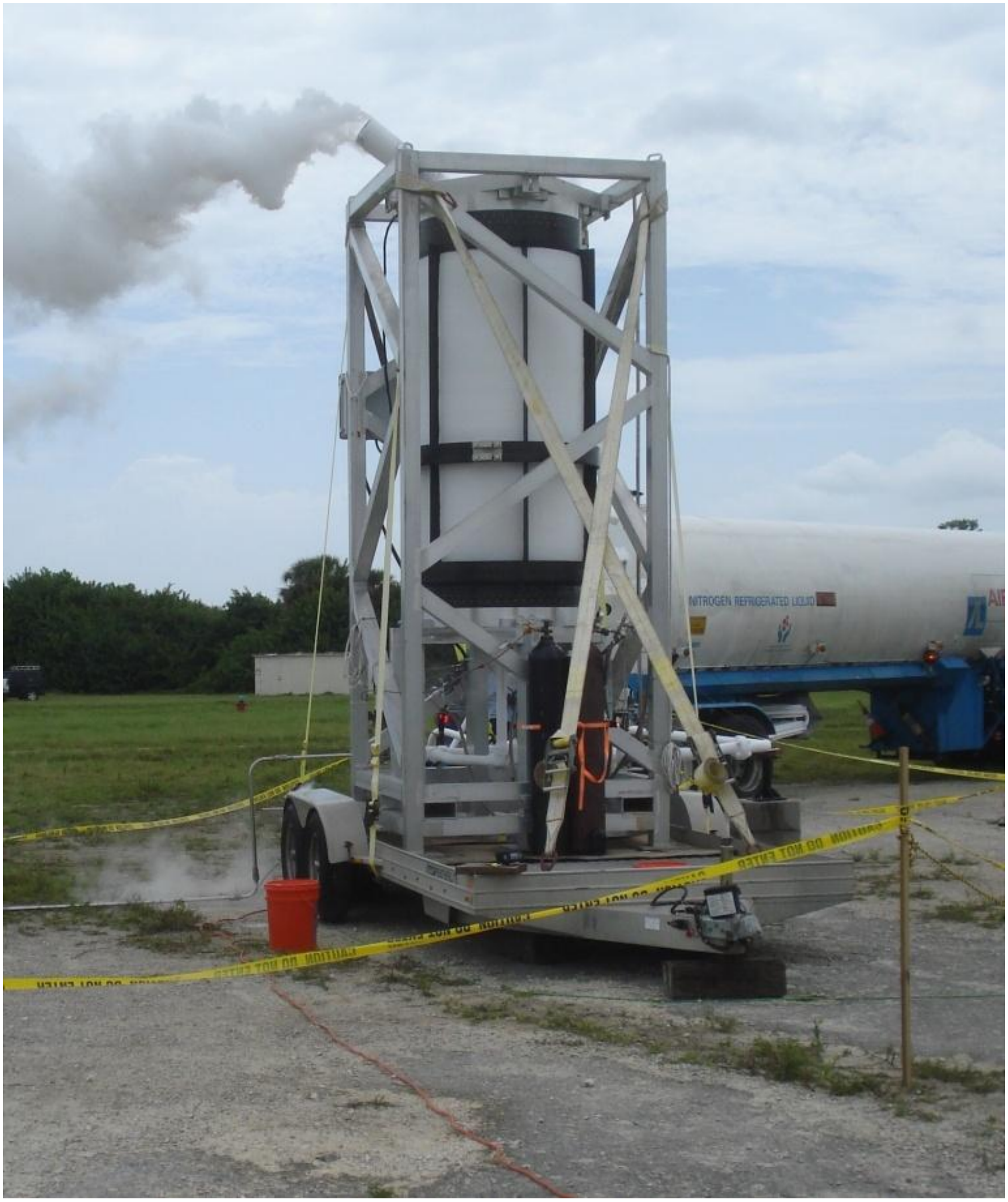


Figure C-12
100% Full, Run 2 (Combined Top and Bottom Fill)

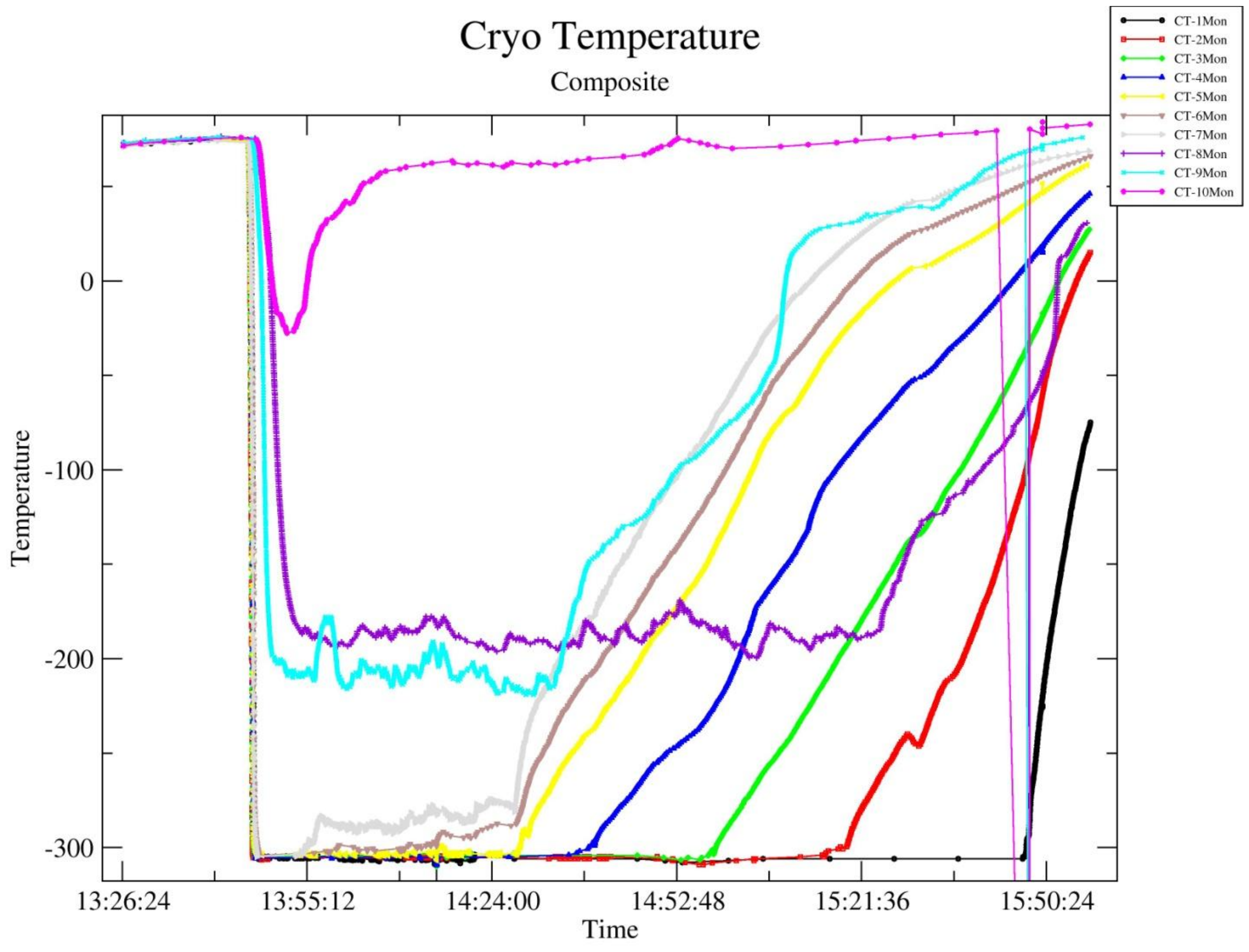


Figure C-13
Composite Temperature Graph, Run 1 (Bottom Fill)

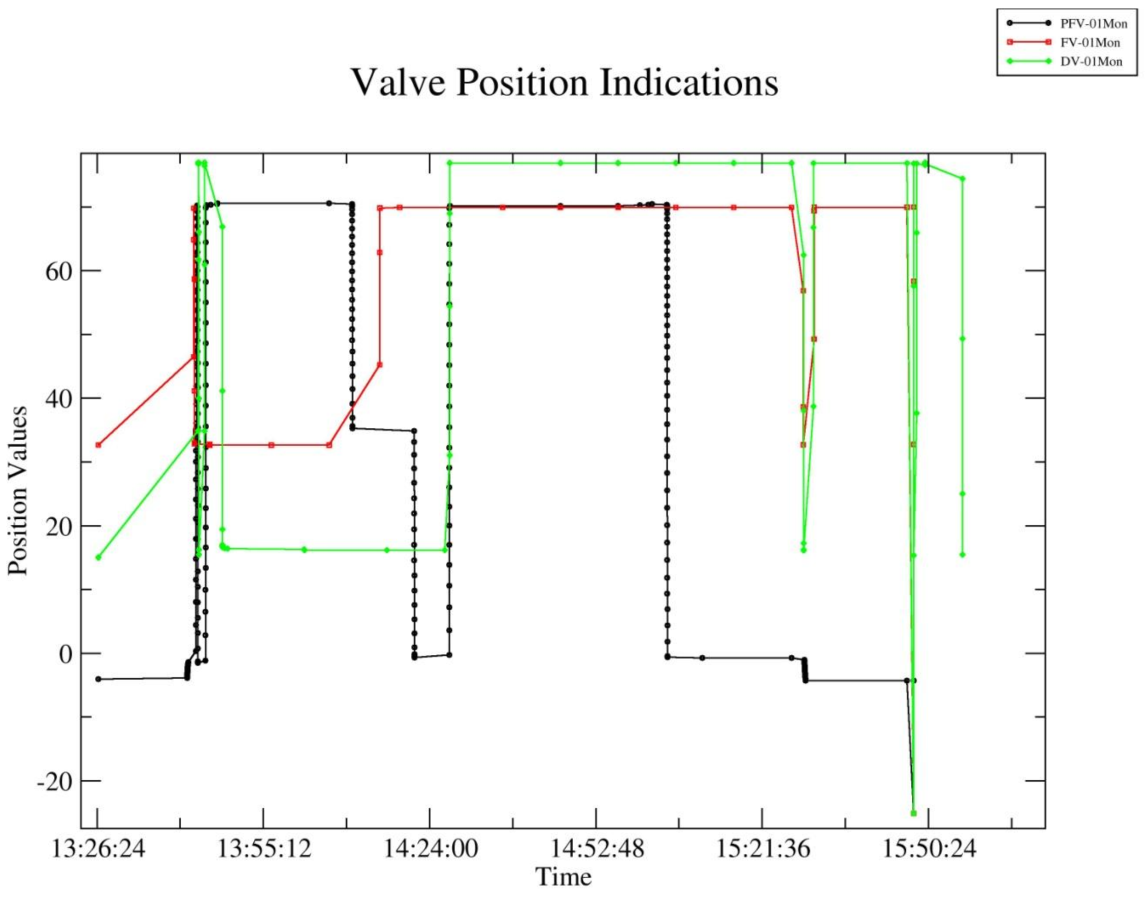


Figure C-14
Valve Position Indicators, Run 1 (Bottom Fill)

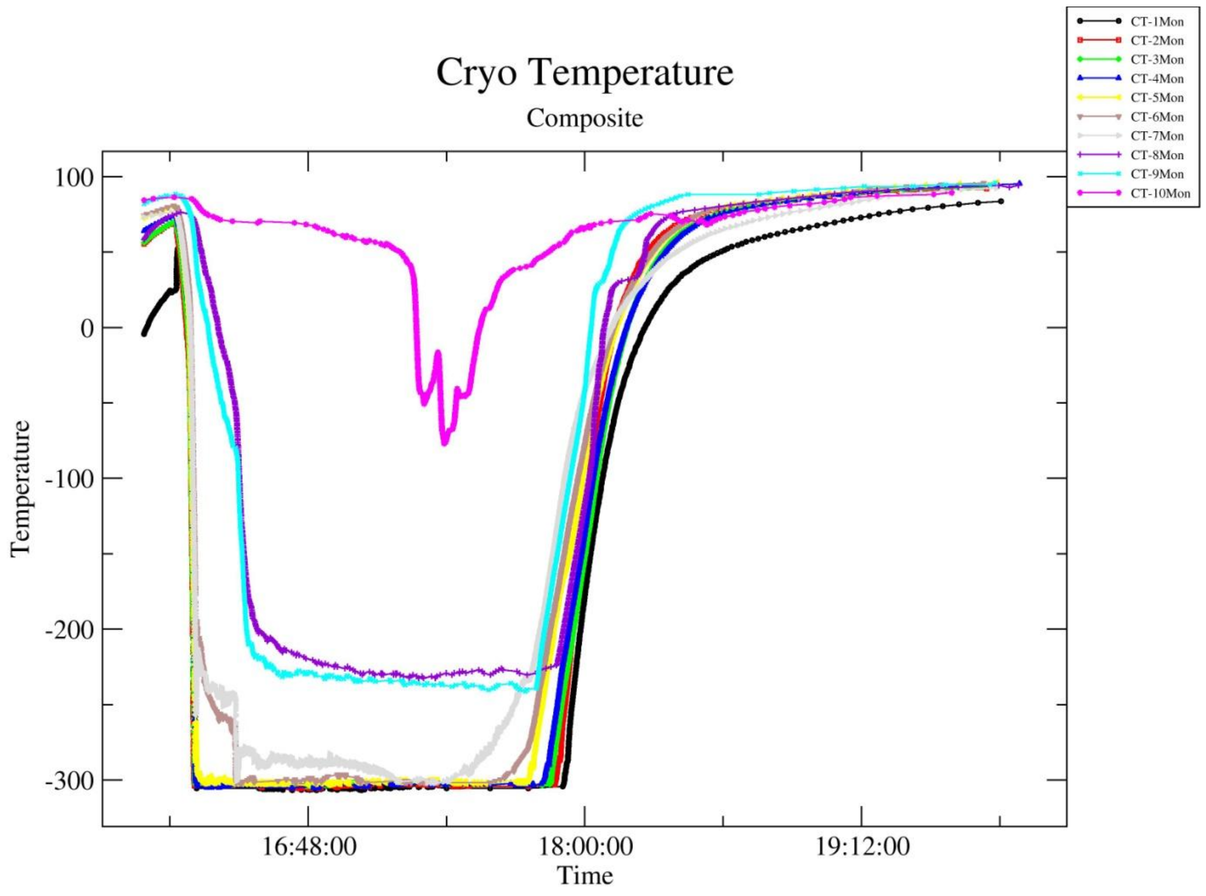


Figure C-15
Composite Temperature Graph, Run 2 (Combined Top and Bottom Fill)

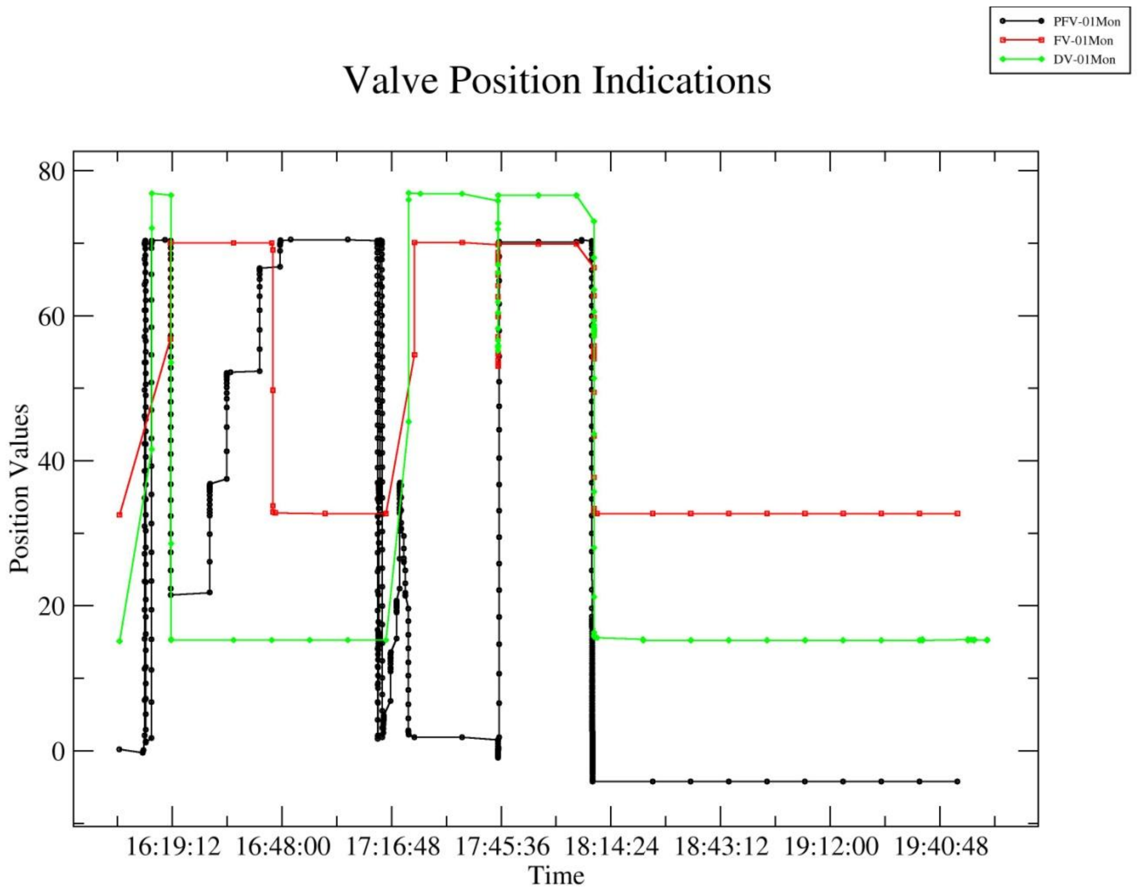


Figure C-16
Valve Position Indicators, Run 2 (Combined Top and Bottom Fill)

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

ACRONYM	**DESCRIPTION**
ADAPT	Advanced Diagnostics and Prognostics Test-bed
AFRL	Air Force Research Laboratory
API	Application
BH	Bulkhead/Connection Point
C&C	Command and Control System
CLLS	Capacitive Liquid Level Sensor
CP	Calibration Port
CT	Temperature Sensor
Degf	Degrees Fahrenheit
DV	Drain Valve
EEAP	Emergency Evacuation Assembly Point
EV	Electronic Valve
FDIR	Fault Detection Isolation and Recovery
FH	Flex Hose
FL	Filter
FM	Flow Meter
FV/PFV	Flow (Control) Valve
GN/ GN2	Gaseous Nitrogen
GOX/GO2	Gaseous Oxygen
HyDE	Hybrid Diagnostics Engine
ICE	Internet Communications Engine
KATE	Knowledge Based Test Engineer
KB	Knowledge Based
KSC	Kennedy Space Center
LAIR	Liquid Air
LI	Latest Issue
LLS	Liquid Level Sensor
LN2	Liquid Nitrogen
LOX/ LO2	Liquid Oxygen
LRU	Line Replaceable Unit
MBR	Model Based Reasoning
MV	Manual Valve
N/A	Not Applicable
NASA	National Aeronautics and Space Administration
NC	Normally Closed
NO	Normally Open
NRHOWG	Non-Routine Hazardous Operations Working Group
O2	Oxygen
OPS	Operations
PGS	Prompt Global Strike
QD	Quick Disconnect
PG	Pressure Gauge
POSU	Pre-operation Setup

PLC	Programmable Logic Controller
PPE	Personal Protective Equipment
Psig	Pounds Per Square Inch (lb/in^2) Gage
PT	Pressure Transducer
PV	Pneumatic Valve
RB	Air Vehicles Directorate of the Air Force Research Laboratory
Reg	Regulator
RPL	Rapid Propellant Loading
RV	Relief Valve
SAC	Security Assistance Corporation
S/O	Shut off
SMS	System Mechanical Schematic
SV	Solenoid Valve
TBD	To Be Determined
UPS	Uninterruptible Power Supply
VLV	Valve